④

RADC-TR-89-337
Final Technical Report
January 1990

SOFTWARE TECHNIQUES FOR NON-VON NEUMANN ARCHITECTURES

Computer Sciences Corporation

Chris Lightfoot, Doug Sakal, Tim Busse, Jerry Shelton

AD-A220 390

DTIC
ELECTE
APR 04 1990
D

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

90 04 04 125

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

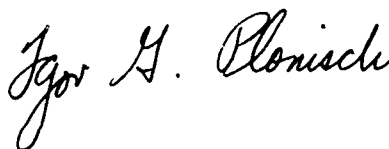RADC-TR-89-337 has been reviewed and is approved for publication.

APPROVED: *[signature]*

PAUL M. ENGELHART
Project Engineer

APPROVED: *[signature]*

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: *[signature]*

IGOR G. PLONISCH
Directorate of Plans & Programs

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release;
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-337

6a. NAME OF PERFORMING ORGANIZATION Computer Sciences Corporation	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c. ADDRESS (City, State, and ZIP Code) 3160A Fairview Park Drive South Falls Church VA 22042		7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center	8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-87-D-0092

8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO. 62702F	PROJECT NO. 5581	TASK NO QB	WORK UNIT ACCESSION NO. 03

11. TITLE (Include Security Classification)

SOFTWARE TECHNIQUES FOR NON-VON NEUMANN ARCHITECTURES

12. PERSONAL AUTHOR(S) Chris Lightfoot, Doug Sakal, Tim Busse, Jerry Shelton - CSC; Ralph Duncan - Control Data Corporation; Tom Cheatham - Software Options, Inc.

13a. TYPE OF REPORT Final	13b. TIME COVERED FROM Feb 88 TO Jun 89	14. DATE OF REPORT (Year, Month, Day) January 1990	15. PAGE COUNT 222

16. SUPPLEMENTARY NOTATION

N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Parallel Processing Non-von Neumann Architectures
12	05		Software Engineering Command and Control

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report examines the use of non-sequential (Non-von Neumann) technology for the support of command and control applications. It has become apparent that the utilization of traditional von-Neumann computers is insufficient to handle the increasing complexity of many applications. The utilization of Non-von Neumann Architectures is needed to satisfy these computational requirements.

In order to assess the utility of Non-von Neumann Architectures, three tasks were performed: 1) A comprehensive survey of existing Non-von Neumann Architectures and development of a new taxonomy, classifying Non-von Neumann Architectures according to structure and capability; 2) Based on the architecture survey and classification scheme, a determination of how these architectures are currently applied as well as their potential use in C3I applications was made; 3) An assessment of how these Non-von Neumann based architectures can be utilized over the entire system and software life cycle.

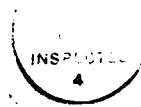| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Paul M. Engelhart	22b. TELEPHONE (Include Area Code) (315) 330-4476 22c. OFFICE SYMBOL RADC (COEE)

DD Form 1473. JUN 86   Previous editions are obsolete.   SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

This report examines the use of Non-von Neumann (NvN) technology for the support of command and control applications. With the introduction of increasingly sophisticated sensors, data input to the command and control function is rapidly becoming too great for standard von Neumann computers to process. Applications such as signal processing, threat assessment and weapon selection and control all need greater processing power to keep up with the modern data acquisition capabilities.

In order to assess the utility of NvN technology, the report defines a taxonomy of seven classes of architectures, the applications for which they are appropriate and the tools available for constructing software. The goal is to provide an indication of what is reasonable to expect from such architectures today and what is required to make the architectures more useful in the future. The taxonomy provides a hierarchical classification structure and emphasizes differences between architectures that otherwise share common high level features. This allows highlighting of features that may determine suitability for particular applications.

The utility of the NvN architectures is considered in the context of Battle Management/Command, Control, Communications and Intelligence (BM/C^{3}I) applications. Existing, operational BM/C^{3}I systems, based on the traditional von Neumann architecture are hard-pressed to cope with the explosion of information that is required by command authorities to successfully manage modern weapons on battlefields of global or near global scope. Life cycle support agencies are responsible for providing computer-based BM/C^{3}I systems to the nation's combat units. Their challenge in the 1990's will be to apply the products of on-going NvN research and development to existing and near-term planned BM/C^{3}I systems.

The existence of an application on a computer system can be attributed to one of two reasons: 1) the architecture was specifically designed to solve problems in that applications' domain or 2) the hardware was available so the application was ported to it. In either case, the efficiency of the application is also dependent on the mapping of the algorithm to the architecture and the construction of the software. As a result, one of the most important areas for immediate consideration is the development of tools and environments for the efficient construction of the software for these high powered machines.

The report considers software tools and environments in the context of software development requirements across the spectrum of NvN architectures. Specific tools, such as vectorizing FORTRAN compilers are typically made available by machine manufacturers as part of the commercial hardware package. In addition, several tools are available through research institutions, such as performance monitors, debuggers, and simulators.

Over and above the need for tools is the need for total life cycle environments based on techniques appropriate for NvN architectures. Once a machine type has been tentatively selected, the entire development cycle including modeling, prototyping, production development and maintenance requires support, if a quality product is to be delivered to the field. While a generic development environment, suitable to multiple architectures should certainly be considered, it is unlikely, with the extreme differences in architecture being proposed, that "one size fits all". Either the effort and expense of providing suitable, different development environments must be considered or the availability of a development environment for a particular architecture may drive machine selection, rather than the performance capabilities of the architecture.

Taxonomy of NvN Architectures

In order to address the topics of this report consistently, it is important to have a classification scheme which appropriately identifies the various NvN architectures discussed by the report. The growing number of architectural types makes such a classification scheme imperative.

The first part of this report defines a practical taxonomy which is a description of such a classification scheme. The basis of the taxonomy is to provide a hierarchical classification scheme with detailed subcategories. Within each subcategory the individual features of the architectural type are highlighted. This highlighting emphasizes differences among existing or planned architectures that may share many common high level features. The hierarchical structure allows the taxonomy to be easily expanded as new architectures are introduced.

The taxonomy provided in the report follows the categories shown in Figure ES-1. A description of each of the major architectural classes is provided, accompanied by a definition of major subclasses and subcategories, each with its defining attributes. Example survey data for the various subcategories is presented to ensure the reader's understanding of the features of the machines considered to be in each class.
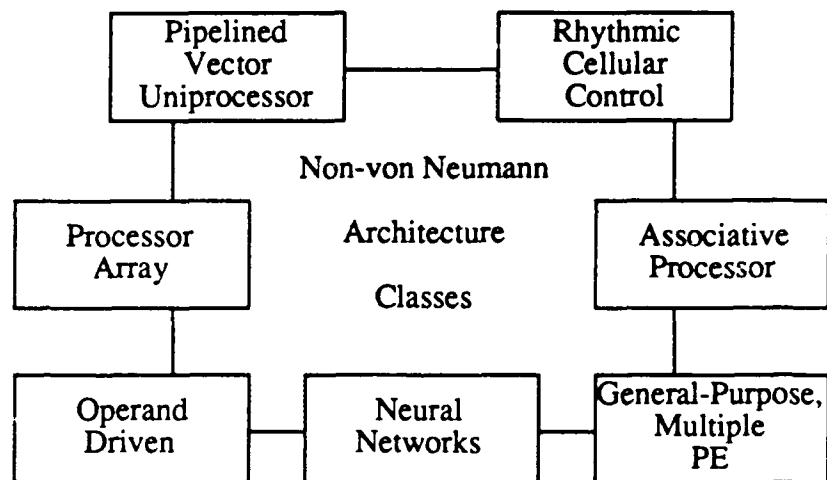
Figure ES-1. The Top-Level Classes of NvN Architectures

The taxonomy can be used to classify new architectures; it can be used as a basis for assessing the strengths and weaknesses of the various architectures with respect to particular problems. Finally, the taxonomy can be used to categorize the environmental support required for proposed machines in order to perform system development in an efficient way.

The taxonomy can be extended by adding new classes in the event of a major architectural breakthrough or by the addition of subclasses and detailed breakouts within subclasses when attributes are provided which do not change the primary operational concept of the machine in question. Maintenance of the taxonomy will provide a current view of the state of NvN machines and the differing capabilities available in the marketplace.

Application Considerations

The challenge to the life cycle support agencies which are responsible for providing computer-based BM/C³I systems to combat units is how to apply the results of on-going research and development of the new NvN machine architectures to existing and near-term planned BM/C³I systems. If there is a single message arising from the many research and development projects investigating the NvN machines, it is that the systems of the 1990's are going to be combination of NvN and traditional machines.

As a basis for analyzing applications and evaluating NvN architectures for use in various problem domains, CSC gathered information application areas where NvN architectures are being used, analyzed the data and extrapolated it to use in future applications. An emphasis was placed on BM/C³I and the current relationship between NvN architectures and BM/C³I problem domains.

Computers that embody NvN architectures potentially offer the computational power required to run many applications in the BM/C³I problem domains. A summary of the domains considered, potential NvN applications and currently unresolved issues is shown in Figure ES-2.
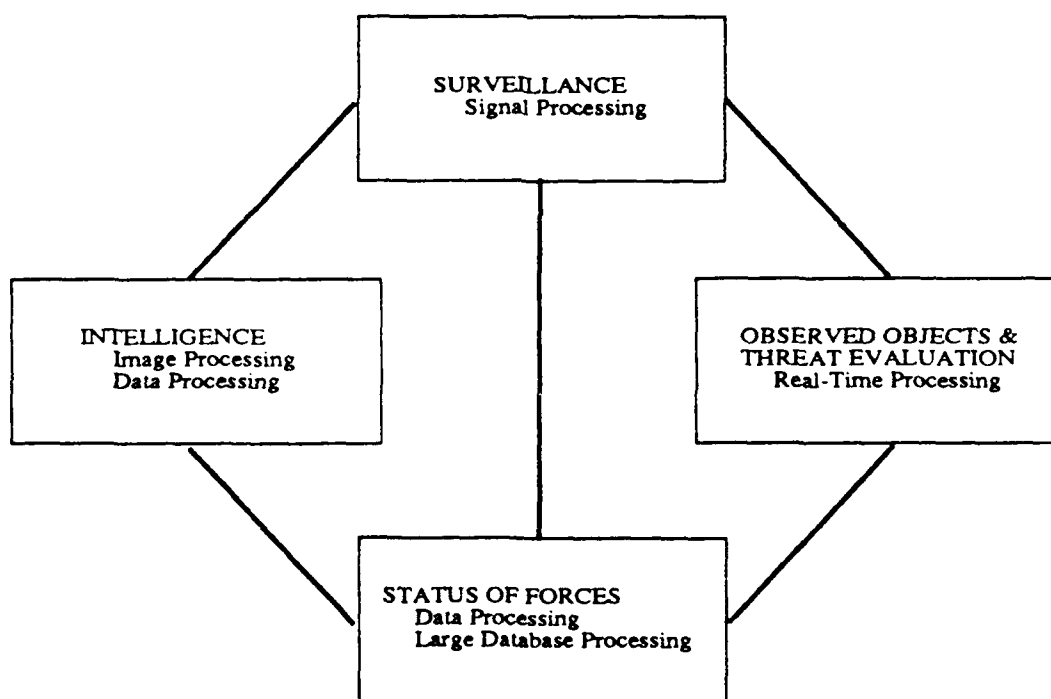
Figure ES-2. BM/C3I Systems in the 1990s.

Additionally, a principle factor in determining applications performance is the algorithm selected for solving components of the problem. An inappropriate algorithm impedes the potential of a computer more than any other factor in determining performance. The capability of a developer to select/design an algorithm for an available architecture or select an architecture suitable for an available algorithm determines the final performance characteristics of the combined system.

Outstanding Issues

Performance levels required for real-time processing are constantly increasing with a need to process and fuse greater amounts of data. The enhanced performance capabilities of the NvN architectures are a natural solution to the data processing requirement.

Size and scope of databases are constantly increasing. A natural approach in this area is the application of the various associative memory machines which are becoming available. Other network-oriented, data base mechanisms also upgrade future database processing.

Imbedding and utilizing human knowledge is becoming an understood technique. However this too requires access to considerable computing power in order to provide decision support capability in real-time. NvN architectures are already being heavily used for pattern recognition. It can be assumed that they will contribute heavily in the development of expert decision support systems in the near future.

Software Engineering Assessment

As the influence of software has become predominant in the information system industry, it has been recognized that the basis of sound software development is an adequate set of support tools. For traditional von Neumann computers, tools are needed to support each phase of the development life-cycle from the original requirements analysis through deployment and maintenance in the field. An integrated set of such tools, appropriate across the entire life-cycle, constitute a software development environment.

For the different classes of NvN machines, a software development environment can be defined. The components of the life-cycle remain constant independent of a particular methodology whether they are performed in a straight line (water fall) fashion, continually iterated (spiral) or performed in a high level, informal way followed by a more formal production cycle (prototyping). At some stage of these or any other life-cycles being considered, requirements analysis must be performed, design must be performed, code must be implemented and after integration and deployment the code must be maintained.

Although the performance characteristics of the NvN machines provide the systems designer with an important tool for enhancing systems capabilities, the complexity of the NvN architectures introduce increased complexity into the software development process. This increases the necessity for a set of support tools which can operate through the entire life-cycle, aiding the system designer in selecting the architecture to be used and also aiding the software developers during the development and maintenance. Unfortunately, outside of the specific implementation portion of the life-cycle, few support tools are available. The situation is summarized in Table ES-1.

The conclusion of this study is that the entire area of support environments for NvN architectures needs considerable work to support the use of these architectures for operational systems. The tools which are developed need to be system oriented, rather than specifically developed by machine manufacturers to support their particular architecture. An important consideration at each end of the development process is the capability to rehost applications from one architecture to another for both testing and greater operational efficiency.

Table ES-1. Support tools available for the software life cycle.

| | | LIFE CYCLE PHASES | | | | |
		REQUIREMENTS ANALYSIS	DESIGN	IMPLEMENTATION (CODE & DEBUG)	TEST	MAINTENANCE
T	Operating Systems			X		X
O	Optimizing Compilers			X		
O	Programming Languages			X		
L	Debuggers			X		
S	Performance Monitors			X	X	
	Programming Models	X	X	X		
	Hardware Simulators	X	X	X		

CHAPTER I. INTRODUCTION

1.1 THE NvN ARCHITECTURE CLASSIFICATION SCHEME (NvNACS)

The NvNACS is based on the following principles:

- the classification scheme is hierarchical

- high-level categories reflect aggregate architectural features wherever possible

- the scheme builds upon several earlier taxonomies

- the categories included reflect existing or definitely planned architectures.

The four tiers of the classification hierarchy in descending sequence are (1) Class, (2) Subclass, (3) Order, and (4) Family.

The NvNACS provides for seven instances of the Class category:

Class I	Pipelined Vector Uniprocessor
Class II	Rhythmic Cellular Control
Class III	Processor Array
Class IV	Associative Processor
Class V	Operand-Driven Processor
Class VI	General Purpose, Multiple-Processing Elements
Class VII	Neural Network Processor

1.2 APPLICATION OF NvN ARCHITECTURES TO BM/C^{3}I APPLICATIONS

1.2.1 BM/C^{3}I

Battle Management, Command, Control, Communications, and Intelligence (BM/C^{3}I) Systems are being analyzed as a first step in transitioning them onto the next generation of hardware architectures. Existing, operational BM/C^{3}I systems, based on the traditional von Neumann (TvN) machine architecture, are hard pressed to cope with the explosion of information that is required by command authorities in order to successfully manage modern missile-type weapons on battlefields of global or near-global scope.

The challenge to the Life Cycle Support Agencies which are responsible for providing computer-based BM/C^{3}I systems to the combat units is to determine how to apply the fruits of on-going research and development of the new Non-von Neumann machine architectures to existing and near-term planned BM/C^{3}I systems. If there is a single message arising from the many research and development projects investigating the NvN machines, it is that the BM/C^{3}I systems of the 1990s are going to be networks of hardware and software of perhaps all seven of the NvN architectures, operating in conjunction with machines of the traditional von Neumann type.

1.2.2 Artificial Intelligence

The technology of artificial intelligence and particularly production systems will become increasing important to BM/C^{3}I systems. Production systems will form the backbone of many different expert advisors that will be inserted into new or re-implemented Command and Control applications over the next decade. The nature of modern battle necessitates various expert advisors to support commanders and their staffs in efficiently managing the military resources for which they are responsible. Section 3.2 discusses the potential for implementing production systems on NvN machines. A production system is a rule-based program that iteratively evaluates sets of conditional rules and acts on the results of the evaluation. Production systems are comprised of working memory, a set of rules, and a program that evaluates the rules based on the current state of working memory. Expert systems are production systems that contain rules derived from human experts. Computerized expert advisors in BM/C^{3}I systems will contain rules that encapsulate the domain expertise of commanders and their staffs, as well as various classes of technologists.

The fundamental processes that comprise all production systems are initialization, condition evaluation, rule selection, conflict resolution and rule firing. The critical processes that determine performance on von Neumann computers are condition evaluation and rule firing. Memory access and complex comparison instructions limit the performance on conventional systems. For some large real-time applications, data acquisition requires fast data input capability and data preprocessing prior to information being stored in working memory. Response time is critical in real-time applications and inferences must be made in a short time frame.

Memory subsystem speed is likely to be the critical factor in determining the performance of a production system, because matching production preconditions to the current working memory contents consumes the vast majority of compute time. This implies that a NvN architecture is needed that balances memory access and conditional evaluation. This aspect has encouraged approaches using both associative memory processors and subtrees of low-capacity processors with private memory. Present research suggests that several NvN architecture types can be efficiently exploited for parallel production system execution.

1.2.3 Real-Time Simulation

Real-time simulation enables commanders and their staffs to play "what-if" games in applying various configurations of military resources to different battle scenarios. The real-time simulation reported here is an in-depth look at a specific Air Defense Initiative (ADI) problem. The ADI Technical Evaluation Facility (TEF) models the North American Air Defense environment and provides for interaction between simulated real-world objects and the simulated effects. The characteristics of this complex model are found in most real-time simulations.

The ADI TEF simulation is comprised of several separate models that are controlled by and communicate through a simulator executive. The TEF executive is a hybrid that combines event-stepped simulation with time-stepped simulation, thereby providing a centrally controlled discrete event simulation with an underlying selectable time period.

The simulator executive is the key to a successful simulation and, therefore, should be carefully designed with particular attention given to simulation efficiency and repeatability. For this simulation 10 to 18 minutes is acceptable turnaround time for simulating 11 one-hour time intervals. Examination of the most compute-intensive model revealed processing requirements in excess of 67 MIPS on a von Neumann computer. Moreover, the computer system needed access to

over 33 MBytes of real memory, and over 2 GBytes of on-line data storage. This large amount of data access and data movement is characteristic of most simulation applications.

Each of the individual models for object motion, sensor detection, or environmental calculations are possible candidates for parallel processing. The calculations performed are identical for all objects of the same category, and simultaneous evaluation offers the potential for greatly increased efficiency. For the simulation executive, feasible parallel execution might be the distribution of functions, provided the simulation is repeatable (i.e., executing the simulation with the same input parameters and data result in identical output data). A coarse-grain parallel architecture provides the best choice for the execution, with each large processor having the ability to execute fine-grained parallel calculations, such as vector or array processing.

1.2.4 Signal Processing

Signal processing is the application of algorithms to sampled data from single or multiple sensors for the purpose of extracting intelligence from the data and/or improving the quality of intelligence that may be extracted. Signal processing techniques are applied to many types of signals including: telecommunication, radar, video images, acoustic, seismic, and medical instrumentation.

The processing algorithms are applied for a variety of purposes, such as: improvement of signal-to-noise ratio, speech recognition/speech compression, detection of events, pattern recognition, parameter measurement, and image processing.

The most pervasive problem of signal processing is its computational intensity. In some cases relatively high I/O bandwidths are also required, but computational bandwidth is the predominant problem.

The problem of high data rates from a large number of sensors is aggravated by the additional requirement for high precision computation when using the more sophisticated processing algorithms. Advances in signal processing over the past three decades have brought increasing complexity of the algorithms, ranging from filtering to spectral analysis to adaptive beamforming. These changes in algorithmic complexity have altered the computational load from a factor of N to a factor of N^2 to a factor of N^3 (where N is the number of data samples to be processed in a given time period). In most signal processing applications, the processing load must be handled in "real-time."

A common and significant attribute of most signal processing applications is the use of complex mathematical techniques such as FFT (fast Fourier transform), IIR (infinite impulse response) filtering, FIR (finite impulse response) filtering, and matrix operations. This algorithmic commonality makes it feasible in many instances to select or to design a system architecture that is suitable for multiple signal processing applications.

NvN architectures are already in use in most of the signal processing applications where computational bandwidth requirements indicate the need, and where cost allows. Numerous pipelined array processors (not to be confused with processor arrays) of the class 1 type have been commercially available as peripherals to main-frame computers, and have been applied to many signal processing applications since the early 1970s.

Adaptive beamforming in radar, sonar, and seismic applications has been performed using rhythmic cellular architectures as well as processor array type architectures. Target tracking

applications have also been performed on associative processor architectures. Processor arrays have also been applied to speech and image processing. Various multiple processing element (PE) architectures have been applied to general signal processing, including the application of expert systems technology to signal analysis.

1.2.5 Image Processing

Image processing has been defined in terms of two categories of processing by S.Y. Kung in his book entitled "VLSI Array Processors", Prentice Hall, Englewood Cliffs, N.J., (1988) pp.538. The research activities dealing with images are divided into two disciplines: image processing and image analysis. Image processing consists of enhancement, restoration, reconstruction and coding, etc. Image analysis, on the other hand deals with extraction of lines, curves, and regions in images, classification of objects, texture analysis, analysis of moving objects, and scene analysis. Most image processing tasks are very time consuming. For example, low level operations, such as filtering or enhancement, typically require the order of some tens of machine instructions per pixel. A typical image obtained from a LANDSAT earth resources satellite is about 1000 x 1000 pixels/image. This implies a computation requirement of some tens of millions of instructions per image, not including the computation for any substantive higher level processing. If such simple low level operations are to be performed at a video rate, say 25 to 30 frames per second this means a throughput requirement of about a billion instructions per second. In general, most real-time image processing throughput rates outstrip current parallel architectures. Thus image applications processing have long been (and will continue to be) a major driving force in the development of faster and more powerful parallel machines.

1.3 SOFTWARE ENGINEERING ASSESSMENT

1.3.1. *Software Life Cycle Issues and NvN Architectures*

The complexities of NvN machines and the architectural complexity of the information processing machine clusters that will characterize future BM/C^{3}I systems is such that the traditional waterfall life-cycle model and its specify-before-building paradigm is an inappropriate development template for developing BM/C^{3}I applications.

The System/Software Engineering Environment (S/SEE) will be the primary instrument for supporting all life-cycle activities, from concept modeling to test and evaluation of implemented machine code. In addition the S/SEE will be used by the responsible Life-Cycle Support Agent in the development of evolutionary upgrades to all deployed BM/C^{3}I systems.

1.3.2. *Software Engineering Technology Issues*

Information about the software tools that have been or are being developed, was gathered through technical literature surveys, discussions with vendors, and discussions with users, particularly users in academic research laboratories. The gathered information shows that many tools exist, and that there is much variation in their usefulness.

The basic software development tool for most NvN machines is an operating system (usually UNIX or a variant of UNIX), a FORTRAN compiler, and a loader/linker as well as a run-time support environment. Often, the FORTRAN compilers accept certain extensions to the language that simplify the creation of code segments that can be executed in parallel.

The surveyed tools and tool sets are also discussed in the context of the NvNACS, giving for each architecture class an analysis of existing tools followed by an analysis of tools that are needed for proper utilization of a particular machine class.

1.3.3. The Automation of Software Development for NvN Architectures

As the software tools become mature, they can be incorporated into a programming environment to automate components of the life cycle phases. Initial programming environments for NvN architectures are beginning to be explored, additional research is needed in the non-implementation phases of the life cycle.

CHAPTER II: NON-VON NEUMANN ARCHITECTURE CLASSIFICATION SCHEME

2.1 OVERVIEW

This section discusses the results of the Non-von Neumann Architecture Survey which constituted Subtask 1 of the Software Techniques for Non-von Neumann Architectures Task.

Subtask 1 included both a survey of current state-of-the-art NonVon Neumann (NvN) architectures and the development of a classification system, or taxonomy, for such architectures.

Section 2.1.2 presents the NvN architecture classification scheme. The presentation of that section is:

Section 2.1.2.1: the classification methodology,
Section 2.1.2.2: the nomenclature of the scheme,
Section 2.1.2.3: top-level classes of NvN architectures,
Section 2.1.2.4: the rationale for the classification scheme.

The Section 2.1.2.4 rationale discussion is divided into:

(1) a discussion of aggregate characteristics,
(2) a definition of a hierarchical classification structure,
(3) the identification of departures from the earlier Hayne's taxonomy,
(4) a discussion of antecedent taxonomies.

Sections 2.1.3 through 2.1.9 discuss the seven instances of the CLASS category. Section 2.1.10 presents details of architectures.

Appendix A contains architecture assessment sketches. Appendix B encapsulates performance data on various instances of NvN architectures. Appendix C is a reorientation of the bibliography to show a mapping of architectures to literature citations.

2.1.1 The NvNACS and the Survey of NvN Architectures

The NvN Architecture Survey was undertaken to provide a sound basis for constructing a Non-von Neumann Architecture Classification System (NvNACS). The NvNACS is designed to be a practical classification system, or taxonomy, that can be used to identify computer architectures that are best suited for particular military applications.

The NvNACS differs from the many computer architecture taxonomies proposed by university researchers because their goals for their taxonomies are different from the goals set for this project.

For example, academic classification schemes are sometimes shaped by being developed:

- in conjunction with a formal descriptive notation for computer architectures (e.g., [Hockney and Jesshope 1981])

- to facilitate architecture comparisons, which use a generic architecture to represent an entire group of quite varied machines (e.g., [Flynn 1972]).

The goal for the NvNACS is to help correlate specific military applications to suitable computer architectures. This necessitates a taxonomy that:

- provides a hierarchical classification structure with detailed subcategories

- emphasizes differences between existing or planned architectures that otherwise share common high-level features.

The NvNACS emphasis on detailed lower-level categories and an informal nomenclature is meant to highlight individual architectural features that may determine suitability for particular applications and to make the system easy for personnel with varied technical backgrounds to use.

2.1.2 The Architecture Classification Scheme

2.1.2.1 The Classification Methodology

The methodology for developing the NvNACS is based on the following principles:

- high-level categories reflect aggregate architectural features whenever possible, in order to increase comprehensibility—this follows the general approach used by Haynes.

- the classification system is hierarchical in order to maintain a systematic character and to facilitate precision in mapping algorithms and application domains to architectural categories.

- the classification scheme builds on previously published taxonomies and subtaxonomies for particular types of architectures (e.g., associative processors, data flow architectures).

- the categories reflect existing or planned architectures, rather than every theoretically conceivable permutation of features, although the scheme readily accommodates future extensions.

2.1.2.2 Nomenclature

The four tiers in the hierarchy of the NvN Architecture Classification System are:

- Class: first-(highest)level category
- Subclass: second-level category
- Order: third-level category
- Family: fourth-level category

If the classification system needs to be refined further, additional lower-level categories can be named, starting at the subfamily level.

2.1.2.3 The Top-Level Architecture Classes

The NvNACS provides for seven instances of the Class category:

1. Class One: Pipelined Vector Uniprocessor
2. Class Two: Rhythmic Cellular Control
3. Class Three: Processor Array
4. Class Four: Associative Processor
5. Class Five: Operand-Driven
6. Class Six: General-Purpose, Multiple-PE
7. Class Seven: Neural Network.

Figure 2-1 shows the top-level classes of the NvNACS.
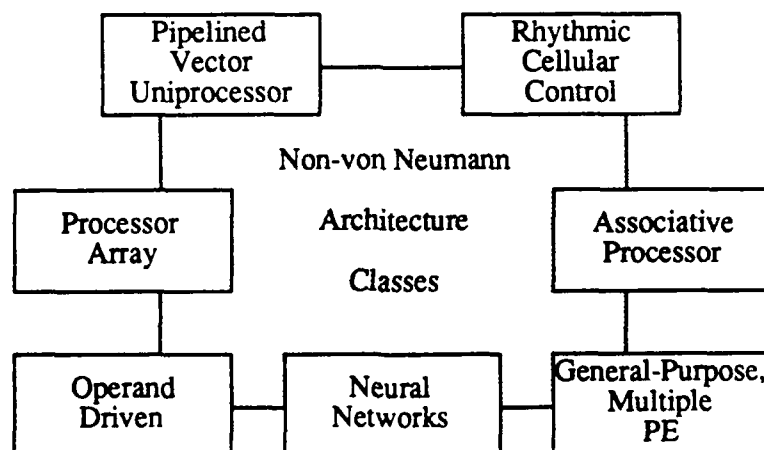
Figure 2-1. The Top-Level Classes of NvN Architecture Classification System

2.1.2.4 Rationale for the Classification System

1. Aggregate Characteristics—Higher-level categories of the NvNACS reflect aggregated architectural characteristics, or features, whenever possible, rather than the presence or absence of a single feature. This approach emphasizes what is distinctive about the overall architecture and the kinds of computational problems it is meant to address. For example, associative memory processors may reasonably be categorized as a type of processor array [Hockney and Jesshope 1981]. However, since associative memory is not an isolated feature of such architectures but rather the central feature which influences the rest of the machine design, the NvNACS provides a separate class for associative processors.

    Constructing categories on the basis of aggregate features that reflect the overall design or a fundamental set of architectural features that naturally go together helps the user to quickly understand high-level classes.

2. Hierarchical Classification Structure—The NvNACS uses a hierarchical classification structure in order to provide an orderly, systematic method for distinguishing architectures that otherwise share important structural or organizational characteristics. Subdividing a class of similar architectures facilitates accurate assessments of architecture suitability for particular algorithms and applications. For example, architectures that share a macroscopic characteristic such as data flow organization may exhibit radically different performance characteristics for a given class of algorithms, due to differences in lower-level implementation characteristics (e.g., expression tree organization vs. packet communications). A hierarchical classification scheme helps expose and organize important architectural differences.

    A hierarchical system, at every classification level, could include categorizations for every possible combination of features. However, this would lead to the creation of many empty categories (those associated with no existing architectures); therefore, this approach has been avoided. Instead, the features of surveyed architectures have been used to produce categories for the combinations of architectural features actually observed in existing or planned machines. Additional NvNACS categories can be added, of course, to reflect future architectural developments.

3. Departures from Haynes Taxonomy—Haynes (1982) originally proposed the following high-level classes of NvN architectures:
    a. Multiple Special-Purpose Functional Units (systolic)
    b. Associative Processors
    c. Array Processors
    d. Data Flow Processors
    e. Functional Programming Language Processors
    f. Multiple CPUs.

Although the NvNACS follows Haynes' approach at the highest level, in that it bases classes on aggregate features, it departs from Haynes' categories in various ways, such as:

a. Multiple Special-Purpose Functional Units has been renamed Rhythmic Cellular Control to more clearly expose the basic organizational principle of these architectures. In addition, the class has been broadened to include wavefront as well as systolic architectures.

b. Following Hockney and Jesshope (1982), Array Processors has been renamed Processor Arrays to distinguish these architectures from commercial products with a single CPU and pipelined functional units, which have been termed array processors.

c. Data Flow Processors and Functional Programming Language Processors have been combined into a single class termed Operand-Driven Architectures.

d. Multiple CPUs has been renamed as General-Purpose, MultiplePE to: (1) make it clear that a broad variety of processing elements can be used in such architectures and (2) to distinguish general purpose architectures from application-specific ones (such as fixed systolic architectures).

e. A Pipelined Vector Uniprocessor class has been added in order to clarify distinctions between this kind of architecture and others, such as MIMD architectures and multi-head vector machines (e.g., ETA-10, Cray X/MP-4).

f. A Neural Network class has been added to reflect recent research activities.

4. Antecedent Taxonomies—The NvN Architecture Classification system has benefited from previous computer architecture taxonomies. Some of the most significant of these are discussed below.

Flynn's categorization of architectures [Flynn 1972] on the basis of instruction and data streams (SISD, SIMD, MISD, MIMD) is still used to describe fundamental architecture characteristics. It was not selected as the basis for this classification effort for several reasons: (a) starting with such highlevel categories would require using many levels of subcategorization, resulting in a cluttered hierarchy that would be difficult to comprehend; and (b) for this study's purposes it is more desirable to emphasize significant architectural features, such as associative memory or cellular organization, at the first level of categorization. However, Flynn's terminology provides a useful shorthand for architectural description, and it is used in the NvNACS.

The taxonomy used by Haynes, et al [Haynes 1982] has influenced this classification effort, especially in its use of aggregate machine characteristics to specify high-level

classes. Comparsons between the Haynes taxonomy and the NvNACS were made in the preceding section.

The NvN Architecture survey effort has examined other overall computer architecture or parallel architecture taxonomies, including those appearing in [Schwartz 1983], [Hillis 1985] and [Hockney and Jesshope 1981] (the latter contains a summary of Shore's 1973 classification scheme).

In addition, the NvNACS is indebted to the subtaxonomies for particular architectural classes that were prepared by the following listed persons: [Treleaven, et. al. 1982], and [Srini 1986] for operand-driven architectures; [Hockney and Jesshope 1981] for processor arrays; and [Yau and Fung 1977] for associative processors.

2.1.3 Class One: Pipelined Vector Uniprocessor Architectures

This class contains single CPU architectures that use special purpose functional units to perform parallel arithmetic operations on vector elements. Such architectures are characterized by a vectororiented instruction set; multiple, pipelined functional units for vector and scalar operations; and a single, fast, scalar CPU [Fernbach 1984], [Hwang 1984], [Kung 1984].

Despite having a single CPU, these architectures are justly regarded as NvN machines, since SIMD vector instructions, pipelining, and multiple functional units all provide parallel execution. Note that vector architectures that involve multiple scalar CPUs (e.g., ETA-10, Cray X-MP/4) or that drive their functional units in lockstep with a broadcast instruction (Burroughs Scientific Processor) are described under other NvN architecture types.

The distinguishing feature of this type of architecture—single instructions that implement vector computations—are typically exploited for scientific and engineering applications, such as fluid dynamics and seismic modeling.

The organization of the Pipelined Vector Uniprocessor class is shown in Figure 2-2.
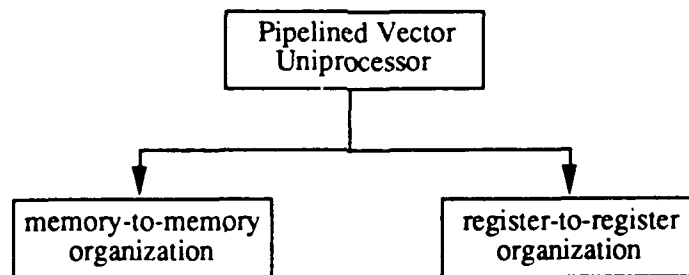
Figure 2-2. Pipelined Vector Uniprocessor Class

2.1.3.1 Pipelined Vector Uniprocessor Architectures: Subclassifications

1. Memory-to-Memory Operation Subclass—Architectures exhibiting memory-to-memory opera-
tion move operands and results directly to and from memory and pipelines.

2. Register-to-Register Operation Subclass—Architectures employing register-to-register opera-
tion move operands and results to a bank of vector registers during transfer operations.

2.1.3.2 Examples of Pipelined Vector Uniprocessor Architectures

Table 2-1 lists examples of pipelined vector uniprocessors.

Table 2-1. Examples of Pipelined Vector Uniprocessors

CLASS: Pipelined Vector Uniprocessor Architectures
SUBCLASS: Memory-to-Memory Operation (1) CDC Star-100 (2) Cyber 205 (3) Texas Instruments Advanced Scientific Computer (ASC)
SUBCLASS: Register-to-Register Operation (1) Cray-1 (2) Fujitsu VP-200 (3) Galaxy (People's Republic of China) (4) Hitachi S-810 (5) NEC SX-2

2.1.4 Class Two: Rhythmic Cellular Control Architectures

This NvNACS category contains systolic and wavefront array architectures, which exhibit rhythmic
cellular control as a principal feature. These architectures differ from more traditional array
processors in at least two significant ways. First, operands, rather than instructions, are broadcast
to PEs (Processing Elements). Second, operands are pulsed from PE to PE in rhythmic fashion. In
the case of systolic architectures the flow of operands is synchronized by a global clock, while
wavefront architectures control operand transmission through asynchronous handshaking.

Rhythmic Cellular Control architectures can be used to implement algorithms that perform regular,
predictable calculations. For example, they are often used for matrix operations involved in signal
processing (e.g., [Kandle 1987], [Nash 1987]). However, programmable systolic arrays, such as
Warp [Annaratone 1986] and the Saxpy Matrix-1 [Foulser and Schreiber 1987], have been
constructed that are not limited to implementing a single algorithm. This class is organized as shown
in Figure 2-3.

```
           ┌──────────┐
           │ Rhythmic │
           │ Cellular │
           │ Control  │
           └──────────┘
                │
        ┌───────┴───────┐
        ▼               ▼
   ┌─────────┐     ┌──────────┐
   │ Systolic│     │ Wavefront│
   └─────────┘     └──────────┘
        │
   ┌────┴────┐
   ▼         ▼
┌───────┐ ┌──────────────┐
│ Fixed │ │ Programmable │
└───────┘ └──────────────┘
```
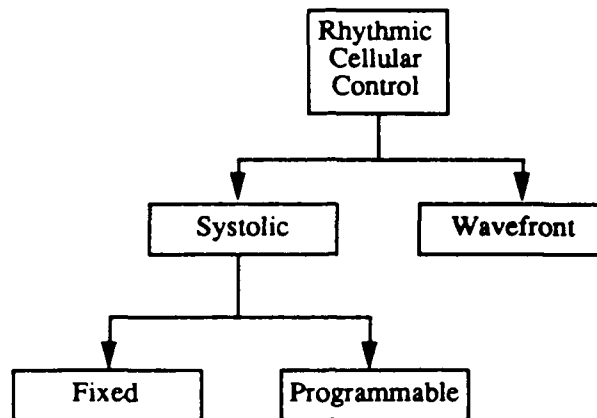
Figure 2-3. Rhythmic Cellular Control Architectures

2.1.4.1 Rhythmic Cellular Control Architectures: Subclassifications

1. Systolic Architectures Subclass—Architectures in the Systolic Subclass have the following characteristics, as described in [Kung 1982], [Kung 1984]:

- data is computed and passed through a network of processing elements in a rhythmic fashion controlled by a global clock for synchronization;

- modular processing elements are united by regular, local interconnections;

- the collection of processing elements executes in pipelined fashion, pumping intermediate results to the next PE for further use;

- systolic systems show execution speed-ups proportional to the number of processing elements [Kung 1984];

- time delays of at least one time unit are used to synchronize operations [Kung 1984];

- only PEs at the boundaries of the PE array communicate with external memory [H.T. Kung 1982].

a. Programmable Systolic Array Architectures Order—These systolic architectures are not algorithm-specific: the functions performed by individual PEs (and often the local connection topology) can be programmed to implement various algorithms.

b. Fixed Systolic Array Architectures Order—These architectures embody a specific algorithm and cannot be changed to implement other algorithms.

II-8

2 Wavefront Architectures Subclass Wavefront architectures significantly differ from systolic architectures in the following ways [Kung 1984]:

- wavefront architectures are asynchronous systems, in which the global synchronizing clock of systolic architectures is replaced by data flow principles (i.e., an operation within a PE takes place when the operands are available);

- the time delays of systolic systems are replaced with asynchronous handshaking between PEs.

2.1.4.2 Examples of Rhythmic Cellular Control Architectures

Table 2-2 lists examples of rhythmic cellular control architectures:

Table 2-2. Examples of Rhythmic Cellular Control Architectures

CLASS....Rhythmic Cellular Control Architectures

SUBCLASS: Systolic Architectures
    ORDER: Programmable Systolic Architectures
        (1) Matrix-1, Saxpy Computer Corporation
        (2) WARP, Carnegie-Mellon University
    ORDER: Fixed Systolic Architectures
        (1) GaAs Systolic Array Beamforming Controller, RCA
        (2) Systolic Adaptive Beamformer, ESL
        (3) Advanced DSP Systolic Array Architecture, Motorola,
        (4) Systolic/Cellular System, Hughes Research Laboratory
        (5) Princeton Nucleic Acid Comparator, Princeton/Brown
        (6) SLAPP (Systolic Linear Algebra Parallel Processor), Naval Ocean Systems Center

SUBCLASS: Wavefront Architectures
        (1) STC-RSRE Wavefront Array Processor System, Standard Telecommunications Company/Royal Signals and Radar Establishment (UK)
        (2) Memory-Linked Wavefront Array Processor, Johns Hopkins Applied Physics Laboratory

2.1.5 Class Three: Processor Array Architectures

Architectures in the Processor Array Class are characterized by multiple processors that cooperatively work in lockstep to perform the same operations on different data elements [Hwang 1984], [Hockney and Jesshope 1981]. Typically, this type of architecture broadcasts a single instruction to all PEs for execution, although some architectures allow individual PEs to disable or modify the instruction. Processor array architectures may be distinguished from commercial array processors, in which a pipelined uniprocessor uses an array of functional units in a pipeline stage, rather than an array of processors ([Hockney and Jesshope 1981], pp. 22-23).

Processor array architectures are commonly used for scientific and engineering applications similar to those found on vector processor architectures. In addition, bit-plane oriented processor array architectures are particularly suitable for image processing applications. Representative applications include: solving partial differential equations, signal processing, weather forecasting, image processing and nuclear energy modeling.

This class is organized as shown in Figure 2-4:
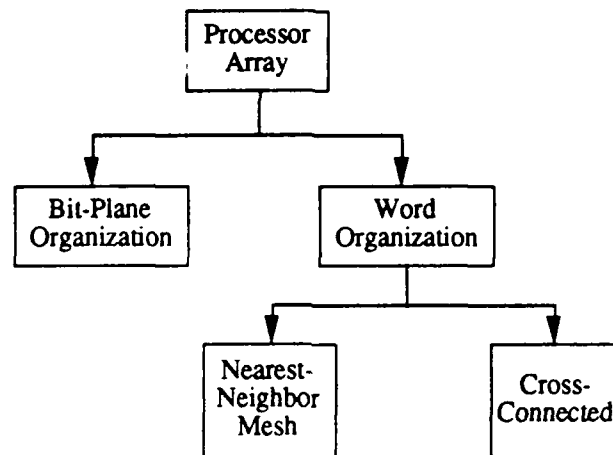
Figure 2-4. Processor Array Architectures

2.1.5.1 Processor Array Architectures: Subclassifications

1. Bit-Oriented Architectures Subclass—This subclass contains architectures that are composed of 1-bit PEs, which work in SIMD fashion. This subclass is further decomposed into orders that reflect whether the PEs' interconnection topology is a grid (mesh) associated with similarly structured memory elements or whether some alternative PE interconnection topology is used.

a. Bit-Plane Oriented Architecture Order—In a Bit-Plane Architecture, an array of 1-bit processors is arranged in a symmetrical grid (e.g., 64x64) and is associated with multiple planes of memory bits that correspond to the dimensions of the PE grid. PE(n), situated in the processor grid at location (x,y), operates on the memory bits at location (x,y) in all the associated memory planes. Usually, operations are provided to copy, mask and perform arithmetic operations on entire memory planes, as well as on columns and rows within a plane.

b. Cross-Connected Topology Architecture Order—This order contains SIMD architectures in which 1-bit PEs are organized in a topology other than a mesh structure. A salient example is the Connection Machine (Thinking Machines Corp.), which organizes 65,536 1-bit PEs (CM-2 model) in a hypercube topology that connects 4x4 PE meshes.

2. Word Oriented Architectures Subclass—This subclass is characterized by PEs that accommodate full word-sized operands, as opposed to the 1-bit PEs subsumed under the Bit Plane Oriented subclass. Operands are often floating point (or complex) values and typically range in size from 32 to 64 bits. This subclass is further subdivided by connection characteristics, following the distinctions presented by Hockney [Hockney 1981].

a. Nearest-Neighbor Mesh Topology Architectures Order—The architectures in this order exhibit comparatively simple mesh-structured connections uniting nearest-neighbor nodes composed of PEs and their associated memories.

b. Cross-Connected Topology Architectures Order—Following Hockney and Jesshope [Hockney 1981], we include in this order all word-oriented processor arrays that exhibit inter-PE connection structures more complicated than a nearest-neighbor scheme.

2.1.5.2 Examples of Processor Array Architectures

Table 2-3 lists examples of processor array architectures.

Table 2-3. Examples of Processor Array Architectures

CLASS: Processor Array Architectures
SUBCLASS: Bit-Oriented Architectures
ORDER: Bit-Plane Architectures

(1) MPP (Massively Parallel Processor) , Loral(Goodyear Aerospace)
      (2) DAP (Distributed Array Processor), ICL
      (3) CLIP4, Imperial College, U.K.

ORDER: Cross-Connected Topology Architectures   (1) Connection Machine, Thinking Machines Corp.

SUBCLASS: Word-Oriented Architectures
   ORDER: Nearest-Neighbor Mesh Topology Architectures
      (1) Illiac IV, Burroughs
      (2) PACS, Tsukuba University,
   ORDER: Cross-Connected Topology Architectures
      (1) Burroughs Scientific Processor (BSP), Burroughs
      (2) GF11, IBM
      (3) Teamed-Architecture Signal Processor (T-ASP), Motorola

2.1.6 Class Four: Associative Processor Architectures

This class contains architectures that are geared to associative memory proce sing and that constitute a distinctive type of array processor. We informally define an associative processor as: (a) accessing

stored data according to its contents and (b) accessing and operating on multiple stored data items through the execution of a single instruction (SIMD operation). The subcategories proposed below follow the work presented in Associative Processor Architecture A Survey [Yau and Fung 1977].

This type of architecture is appropriate for applications that principally involve selecting data base entries in parallel according to their contents. Recorded applications for associative memory processors include tracking and surveillance, cartography, image processing, and signal processing.

The organization of the Associative Processor Class is shown in Figure 2-5:
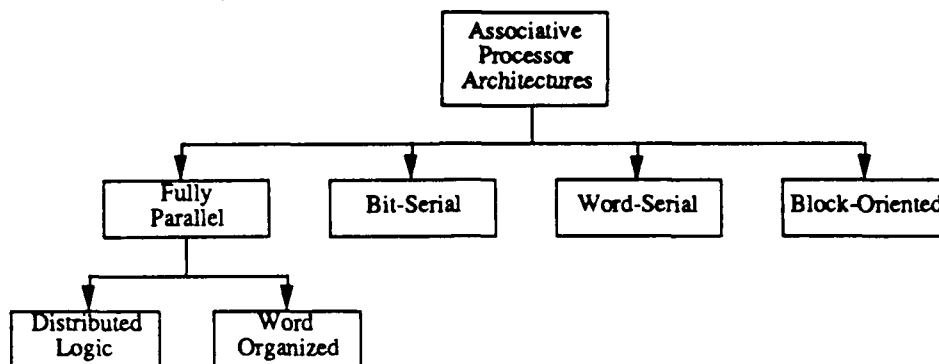
Figure 2-5. Associative Processor Architectures

2.1.6.1 Associative Processor Architectures: Subclassifications

1. Fully Parallel Subclass—In the Fully Parallel Subclass, all bits (or groups of bits) in a given column of memory are accessed by an instruction, and multiple columns can be accessed simultaneously.

    a. Word-Organized Order—Processors in the Word-Organized Order have comparison logic associated with each associative memory bit, and data is available at every memory word's output. The column of memory accessed concurrently is 1 bit wide.

    b. Distributed Logic Order—The Distributed Logic Order differs from the Word-Organized Order in that the columns of concurrently accessed memory are several bits wide, and typically contain enough bits to constitute a character.

2. Bit-Serial Subclass—The Bit-Serial subclass is distinguished by concurrently operating on a single bit-slice (bit-column) of all the words in the associative memory module, but not concurrently operating on multiple bit-slices.

3. Word-Serial Subclass—Essentially, architectures in this subclass use hardware to implement a loop construct for searching.

4. Block-Oriented Subclass—This subclass uses rotating memory devices (e.g., disk) as the

associative memory. It is not clear that this architecture category is currently viable and, therefore, it may be of historical interest only.

2.1.6.2 Examples of Associative Processor Architectures

Table 2-4 lists examples of associative processor architectures.

Table 2-4. Examples of Associative Processor Architectures

CLASS: Associative Processor Architectures

SUBCLASS: Fully Parallel
    ORDER: Word-Organized
    ORDER: Distributed Logic
 	
    (1) PEPE (Parallel-Element Processing Ensemble)

SUBCLASS: Bit-Serial
 	
    (1) ALAP (Associative Linear Array Processor)
    (2) ASPRO (militarized version of STARAN)
    (3) ECAM (Extended Content Addressed Memory)
    (4) OMEN (Sanders Associates)
    (5) RAP (Ratheon Associative/Array Processor)
    (6) STARAN

SUBCLASS: Word-Serial   (1) NEBULA experimental computer (circa. 1964-66)
SUBCLASS: Block-Oriented   (1) RAPID (Rotating Associative Processor for Information Dissemination)

2.1.7 Class Five: Operand-Driven Architectures

Data-Driven (Data Flow) and Demand-Driven (Reduction) architectures are both subsumed under the Operand-Driven Class, since both are characterized by instruction execution that is driven by the status of instruction operands. This common divergence from more traditional architectures militates for placing both kinds of architectures in a common class. Since data-driven and demand-driven architectures employ significantly different mechanisms for triggering instruction execution, they are categorized in different Subclasses.

The organization of this Class is shown in Figure 2-6 on page II-14.

2.1.7.1 Operand-Driven Architectures: Subclassifications

1. Data-Driven (Data flow) Subclass—In data flow architectures, instructions are executed when all of their operands are available; i.e., when any needed computations that reduce expression operands to values have been performed. Multiple processors can handle the instructions as they become enabled for execution.
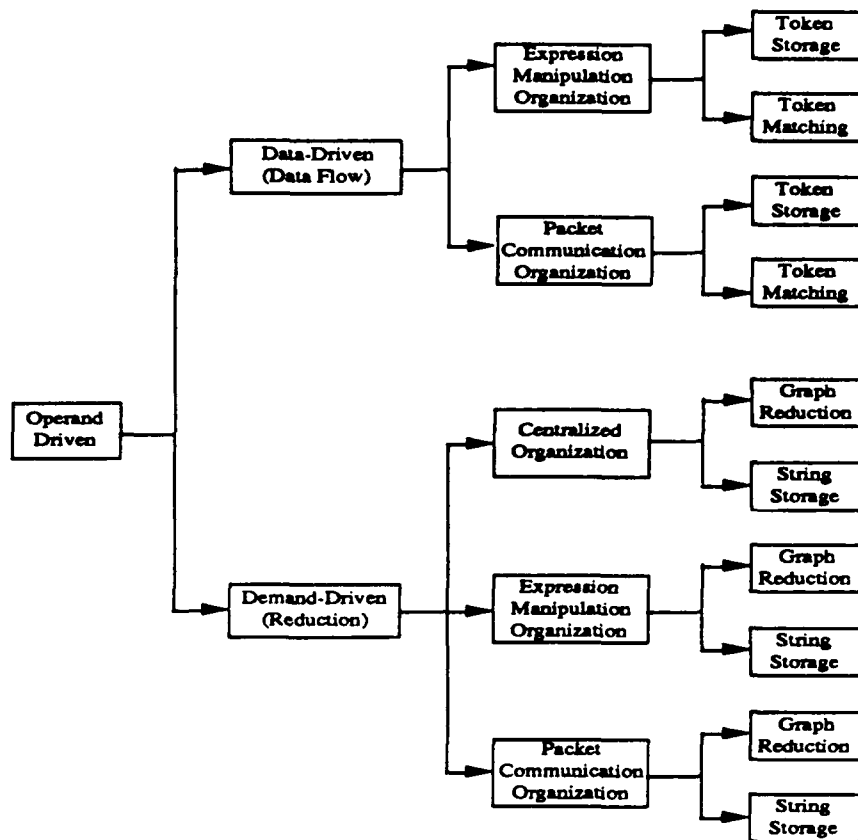
Figure 2-6. Operand-Driven Architectures

2. Demand-Driven (Reduction) Subclass—Demand-driven architectures execute programs in the form of nested expressions, using appropriate rules to order expression evaluation. An instruction is executed when:

- all its operands are available;

- its result is needed as an operand for an instruction that is higher in the hierarchy and that is also slated for execution.

3. An Overview of Machine Organization Order Categories—The Order classification strata beneath both subclasses is based on the tripartite machine organization model that was developed by Treleaven [Treleaven, et al, 1982].

a. Centralized Organization Orders—Centralized organization involves a single processor and single active instruction. It is possible to apply such a traditional architecture to implementing the Demand-Driven (Reduction) subclass with the aid of special hardware and microcoding (GMD, Cambridge SKIM reduction machines), although it is debatable whether this really constitutes an NvN machine.

b. Packet Communication Organization Orders—Packet Communication organization involves having multiple processing elements connected by a circular pipeline. Packets of work (instructions) are distributed among the PEs as operand data becomes available (data-driven subclass) or as the instruction results are demanded for use as operands (demand-driven subclass).

c. Expression Manipulation Organization Orders—Expression Manipulation organization uses multiple nodes, connected in a regular structure (e.g., tree or mesh), where each node has processing, communications and memory capabilities. The structural adjacency of elements in the input program is mapped onto the physical processing node structure.

d. Token Handling Mechanism Families Data-Driven Architectures—For data-driven architectures, the family level of classification is made on the basis of whether token storage or token matching is used.

(1) Token Storage Families (Data-Driven Architectures)

Token storage mechanisms store instruction results (operands for subsequent instructions) in the subsequent instruction as they become available.

(2) Token Matching Families (Data-Driven Architectures)

In a token matching scheme, the execution of an instruction typically produces two kinds of result tokens—data result tokens and control tokens—that specify that a data result will serve as a particular operand in a subsequent instruction. A functional unit matches control tokens to instructions. When a complete set of control tokens (representing all the required operands) is assembled for an instruction, the relevant results are copied from storage into the instruction's operands and the instruction is then executed.

(3) Reduction Mechanism Families (Demand-Driven Architectures)

For demand-driven architectures, family level categorization is made on the basis of whether a string or graph reduction mechanism is used in the evaluation of nested expressions.

(a) String Reduction Families (Demand-Driven Architectures)

The String Reduction mechanism evaluates expressions consisting of literals and copies of values.

(b) Graph Reduction Families (Demand-Driven Architectures)

The Graph Reduction mechanism evaluates expressions that consist of literals and references (pointers) to values.

2.1.7.2 Examples of Operand-Driven Architectures

Table 2-5 lists examples of operand-driven architectures:

Table 2-5. Examples of Operand-Driven Architectures

CLASS: Operand-Driven Architectures
SUBCLASS: Data-Driven
ORDER: Centralized Organization
ORDER: Expression Manipulation Organization
(1) Utah Data-Driven Machine (insufficient data for family subclassification)
ORDER: Packet Communication Organization
FAMILY: Token Matching Mechanism

(1) Irvine Data Flow Machine
        (2) Manchester Data Flow Computer
        (3) M.I.T. Tagged Token Data Flow
        (4) Newcastle Data-Control Flow Computer

FAMILY: Token Storage Mechanism

(1) M.I.T. Data Flow Computer
        (2) Texas Instruments Distributed Data Processor
        (3) Toulouse LAU System

SUBCLASS: Demand-Driven
ORDER: Centralized Organization
FAMILY: Graph Reduction Mechanism
(1) Cambridge SKIM Machine
FAMILY: String Reduction Mechanism
(1) GMD Reduction Machine
ORDER: Expression Manipulation Organization
FAMILY: Graph Reduction Mechanism
FAMILY: String Reduction Mechanism
(1) Newcastle Reduction Machine   (2) North Carolina Cellular Tree Machine
ORDER: Packet Communication Organization
FAMILY: Graph Reduction Mechanism
(1) Utah Applicative Multiprocessing System
FAMILY: String Reduction Mechanism

2.1.8 Class Six: General-Purpose Multiple-PE Architectures

The General-Purpose, Multiple-PE (GPMPE) class contains multiprocessor architectures that fall outside of the other NvNACS classes and that share sufficient application flexibility to merit being termed general purpose. Since GPMPE architectures do not hold a fundamental design approach or hardware feature in common, the class contains architectures that exhibit considerable diversity in

granularity of parallelism, topology, and PE size. Most of these architectures execute in MIMD fashion, although some are capable of (M)SIMD or MIMD/SIMD operation.

In order to provide a systematic subclassification scheme for GPMPE architectures, subclassifications are made on the basis of whether processor-to-processor or processor-to-memory communications predominate, or whether the two are equally fundamental to the architecture. Further subdivisions are based on topology (for processor-to-processor architectures) and interconnection technology (for processor-to-memory architectures). The structure of the GPMPE class is shown in Figure 2-7.
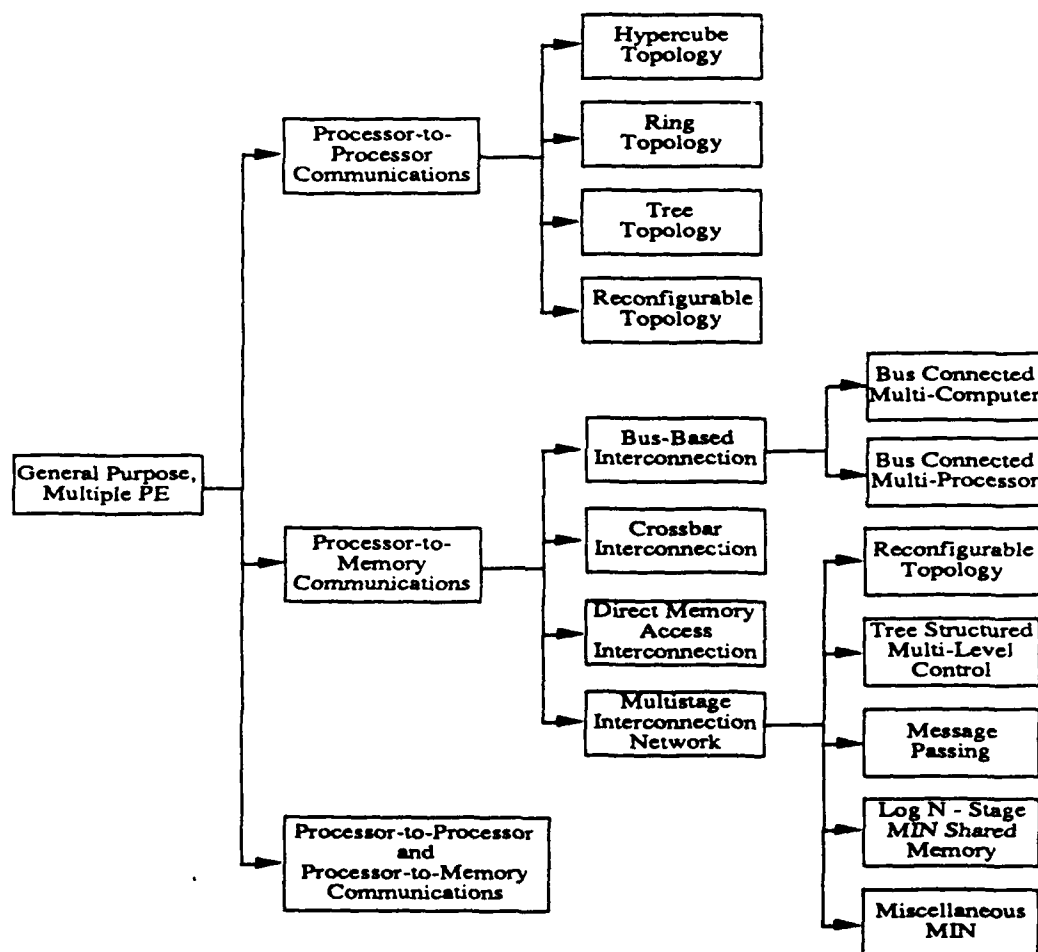
Figure 2-7. General-Purpose, Multiple-PE Architectures

2.1.8.1 General-Purpose, Multiple-Processor Architectures: Subclassifications

1. Processor-to-Processor Communication Architecture Subclass

   a. Hypercube Topology Architecture Order—A boolean n-cube or hypercube topology [Seitz 1985] uses $N = 2**n$ processors arranged in an n-dimensional cube, where each

node has $n = \log_2(N)$ bidirectional links to adjacent nodes. Individual nodes are uniquely identified by n-bit numeric values that range from 0 to N-1 and that are assigned in a manner that ensures that the value of adjacent nodes' values differ by a single bit. Messages contain a destination bit-value and a label initialized to the source node's bit-value. When a processor routes a message, it first selects an adjacent node that has a bit in common with the destination value (which the routing node lacks), then corrects that bit of the message label. As a result of these conventions, the number of links traversed by a message travelling from node A to node B is equal to the number of bits that differ in the two nodes' bit-values; hence, the diameter of the Interconnection Network (maximum bit-value difference) is $n = \log_2(N)$.

(1) Ring Topology Architecture Order—Several GPMPE machines manufactured by Control Data Corporation exemplify the ring topology approach to linking processors. These machines use direct, hardwired connections to link each processor with its two neighbors. Two ring networks are formed in this fashion: one ring transfers data in a clockwise direction, while the other conducts counterclockwise transfers. Fixed-size packets that include a destination address are sent between processors, passing from one PE to another until reaching their destination processor. This ring interconnection differs from a classical bus linkage in that multiple processors may simultaneously use the interconnection network. Some versions of these machines provide a shared memory for the processor nodes to access. Commercial ring topology architectures include the Advanced Flexible Processor, the Cyberplus and the Parallel Modular Signal Processor [Control Data Corp., August 1980, March 1986, and February 1987].

(2) Tree Topology Architecture Order—Tree topologies can readily be exploited to partition some of a system's PEs into a processor set that executes in SIMD mode. The concurrent operation of such processor sets and the remaining processors provides an architecture with MIMD/SIMD and (M)SIMD capabilities. The hierarchical structure inherent in tree topologies facilitates such partitioning, since the master/slave relation of SIMD processor control can easily be mapped onto the node/descendent relation of the tree structure. Communication diameter, however, is a potential problem for tree topologies. For example, a complete binary tree with n levels (and 2n-1 processors) has a communication diameter of 2(n-1).

Proposed solutions to this problem include linking all the nodes at each level [Despain and Patterson 1978] or, as is the case with DADO2 [Stolfo 1987], providing a specialized I/O switch and combinational circuit that effectively links all nodes.

(3) Reconfigurable Topology Architecture Order—Although the components of GPMPE architectures obviously possess an underlying physical topology, one can reasonably designate architectures as reconfigurable topology machines if providing user-programmable interconnection topology is a fundamental aspect of their design. Physical

topologies that have been incorporated in reconfigurable topology architectures include CHIP's mesh [Snyder 1982], [Kapauan, et al, 1984] and TRAC's Banyan [Lipovski and Malek 1987].

It is the macroscopic capabilities these provide that make them reconfigurable topology machines, since architectures offering different high-level capabilities (for example, a high degree of fault-tolerance) can employ similar flexible interconnection techniques [Adams et al, 1987], [Abraham et al, July 1987].

Reconfigurability functions range from specifying different topologies [Snyder 1982], [Kapauan, et al, 1984] to partitioning a base topology into multiple interconnection topologies of the same type [Siegel et al, 1987].

2.   Processor-to-Memory Communication Architecture Subclass

a. Bus Interconnection Architecture Order Bus-based GPMPE architectures (e.g., Encore Multimax, ELXSI System 6400) use one or more buses to give multiple PEs common access to a shared memory. Typically, a moderate number of processors (4-32) are provided. Some bus-based architectures, such as the experimental Cm* system developed at Carnegie Mellon University [Jones and Schwarz 1980], employ 2 kinds of buses a local bus to serve PEs within a cluster, and a system bus that links dedicated service processors associated with each cluster.

b. Crossbar Interconnection Architecture Order GPMPE architectures characterized by crossbar interconnection technology use a crossbar switch of $n**2$ crosspoints to connect n processors to n memories. Although processors may contend for access to the same memory location, crossbar schemes prevent contention for communication links by providing a dedicated pathway between each possible processor/memory pairing. Power, pinout, and size considerations, however, have limited crossbar architectures to a small number of processors (i.e., 4-16).

c. Direct Memory Access Interconnection Architecture Order

The term direct memory access is used here to designate GPMPE architectures in which processors effectively share a global memory by copying an entire memory 'page' in a single parallel operation, rather than by obtaining memory contents through sequential byte or word-level operations.

In practice, DMA access to a shared memory is typically employed by vector processing supercomputers (e.g., the Cray X-MP/4 and ETA-10) with 4-8 processors in order to provide parallelism at the task level.

d. Multistage Interconnection Network (MIN) Architecture Order

These architectures use a multistage interconnection network (MIN) [Bhuyan 1987], [Siegel 1985] to connect processors and memories. A MIN deploys multiple stages or banks of switches in the pathway connecting the processors and memories. A popular approach is to connect n processors to n memories by using log2(n) stages of n/2 switches: where each switch accommodates two inputs and two outputs. Proposed variations of such log2(n) stage MINs include the omega, flip, indirect binary n-cube, SW-banyan, butterfly, multistage shuffle-exchange, baseline, delta, and generalized cube networks [Bhuyan 1987], [Siegel 1985], [Miller 1988]. A significant feature of these architectures is their expandability, accruing from a communication diameter that is proportional to log2(n).

3. Processor-to-Processor and Processor-to-Memory Communication Architecture Sub-class

Although NvN architectures often feature either processor-to-processor or processor-to-memory interconnection networks, both kinds of communication can be provided by a GPMPE architecture. The Texas Reconfigurable Array Computer (TRAC) [Lipovski and Malek 1987] is an example of this hybrid approach.

Reconfigurable topology is an essential feature of TRAC, which allows programmable interconnections linking processors, memories and I/O devices. TRAC provides both circuit and packet switching interconnections.

2.1.8.2 Classifying General-Purpose, Multiple Processors

Table 2-6 on page II-21 shows examples of General-Purpose, Multiple-PE architectures.

2.1.9 Class Seven: Neural Networks Architectures

Neural networks are connectionist architectures [Fahlman and Hinton 1987] characterized by simple PEs linked by an interconnection network. The state of weighted PE interconnections embodies program knowledge, typically a function for mapping inputs to desired outputs.

There are significant differences among existing and proposed neural networks, although there are some common basic organizational principles:

- the behavior of PEs and interconnections reflects a simplified model of biological neurons and synapses.

Table 2-6. Examples of General-Purpose, Multiple-PE Architectures

CLASS...General Purpose, Multiple-PE Architectures

SUBCLASS: Processor-to-Processor Communication Architectures

ORDER: Hypercube Topology Architectures

(1) Ametek Series 2010
(2) Cosmic Cube, California Institute of Technology
(3) Intel Personal Supercomputer (iPSC), Intel Corp.
(4) NCUBE/10

ORDER: Ring Topology Architectures

(1) Advanced Flexible Processor, Control Data Corp.
(2) Cyberplus, Control Data Corp.
(3) Parallel Modular Signal Processor, Control Data Corp.

ORDER: Tree Topology Architectures

(1) DADO2, Columbia University
(2) NON-VON, Columbia University

ORDER: Reconfigurable Topology Architectures

(1) Armstrong Multicomputer, Brown University
(2) Configurable Highly Parallel multicomputer (CHiP), University of Washington
(3) Computing Surface, Meiko (U.K.)
(4) PASM, Purdue University
(5) TRAC, University of Texas

SUBCLASS: Processor-to-Memory Communication Architectures

ORDER: Bus Interconnection Architectures

(1) Cm*, Carnegie-Mellon University
(2) Encore Multimax, Encore Computer Corporation
(3) ELXSI System 6400, ELXSI
(4) FLEX/32, Flexible Corporation
(5) SPUR, University of California, Berkeley

ORDER: Crossbar Interconnection Architectures

(1) Alliant FX/8, Alliant Computer Systems Corp.
(2) S-1, U.S. Navy

ORDER: Direct Memory Access Interconnection Architectures

(1) Cray X-MP/4, Cray Research
(2) ETA-10, Control Data Corp.

ORDER: Multistage Interconnection Network (MIN) Architectures

(1) Butterfly Parallel Processor, Bolt Beranek and Newman
(2) CEDAR, University of Illinois
(3) HEP, Denelcor Inc.
(4) RP3, IBM
(5) Ultracomputer, New York University

SUBCLASS: Processor-to-Processor and Processor-to-Memory Communication Architectures

(1) TRAC, University of Texas

- a PE's output is usually calculated as a function of weighted inputs from other PEs, subject to thresholding [Graf et al, 1988].

- the interconnection network that carries weighted inputs and outputs exhibits a directed-graph topology and is often organized in layers.

- the interconnectivity of the network can be engineered to make individual PEs sensitive to the global network state [Fahlman and Hinton 1987], [Graf, Jackel and Hubbard 1988].

- PEs adjust their output calculation rules dynamically—by altering the weights associated with inputs from other PEs—in order to give the system an adaptive character.

Various neural network learning algorithms and paradigms for dynamic weight adjustment have been proposed. Table 2-7 lists seven learning algorithms.

Table 2-7. Neural Network Learning Algorithms

Adaptive Resonance Theory	[Carpenter and Grossberg 1987]
AR-P	[Barto 1985]
Backpropagation	[Rumelhart, Hinton, and Williams 1986]
Competitive Learning	[von der Malsburg 1973]
	[Grossberg 1976]
Counterpropagation	[Hecht-Nielsen 1988]
Hopfield Energy Minimization	[Hopfield 1982]
	[Hopfield and Tank 1985]
Kohonen Learning	[Kohonen 1984]

2.1.9.1 Neural Network Architecture: Subclassifications

The subclassification system for neural network architectures is still at an embryonic stage, because comparatively few machines based on neural network principles have actually been built. Many neural network products are actually software packages that model proposed neural network functioning.

1. Emulation Coprocessor Subclass This subclass consists of coprocessors, typically organized at the board level, which run dedicated software that models neural network functioning. Note that this hardware usually employs von Neumann microprocessors and does not constitute a neural network implementation. These processors are included in the NvNACS to satisfy the completeness criterion.

2. Hardware Neural Network Implementations Subclass The architectures in this class physically embody the neural network characteristics discussed above. This subclass can be further articulated to the Order taxonomic level when the sample of hardware neural networks is large enough to reasonably support further subclassification.

2.1.9.2 Examples of Neural Network Architectures

Table 2-8 shows examples of Neural Network Architectures.

Table 2-8. Examples of Neural Network Architectures

CLASS: Neural Network Architectures
SUBCLASS: Emulation Coprocessors (1) Anza-Plus Neurocomputing Coprocessor, Hecht-Nielsen Neurocomputers (2) Neural Phonetic Typewriter, Kohonen (Helsinki Univ. of Technology
SUBCLASS: Hardware Neural Network Implementations (1) CMOS VLSI Neural Network, AT&T Bell Labs (2) Neural Network Chip, Bell Communications Research (3) Resistive Networks, Koch and Mead (Caltech) (4) Speech Recognition Circuit, Hopfield (Caltech),   Tank and Unnikrishnan (AT&T Bell Labs)

2.1.10 ARCHITECTURE DESCRIPTIONS

This section, containing Non-von Neumann Architecture data is included to show the kind of information that shaped the NvNACS.

Name	Alliant FX/8
Company	Alliant Computer Systems Corp.
Stream	MIMD
Commtech()	circuit-switching crossbar
Commtopo	crossbar
Control	UNKNOWN_
Assign	TBD Memory: SHARED
Synch	UNIVERSAL
Max_cpu	20 (8 vector PEs + 12 interactive processors)
Cpu_size	64
Perform	94.4 Mflops (32-bit vectors); 47.2 MFLOPS (64-bit vectors) [Dongarra 1987]
Market	Engineering and scientific
Software1	Concentrix-OS;Pascal,C;FORTRAN 77 with VAX/VMS extensions
Software2	FORTRAN 8x array extensions; debugger; auto-vectorization and parallel detection
Comment1	Pipelined vector machine
Comment2	

Name	Ametek Series 2010
Company	Ametek
Stream	MIMD
Commtech()	wormhole-routing (hardware routing chips)
Commtopo	2D mesh (with 4x4 submeshes)
Control	DECENTRALIZED
Assign	HYBRID Memory: PRIVATE
Synch	CONDITIONAL
Max_cpu	
Cpu_size	32-bit (Motorola 68020 + microprogrammed queue-manager PE)
Perform	
Market	
Software1	uses Caltech "Cosmic Environment/Reactive Kernel" O.S
Software2	
Comment1	[Athas and Seitz 1988]
Comment2	

Name	Anza-Plus Neurocomputing Coprocessor System
Company	Hecht-Nielsen Neurocomputer
Stream	
Commtech()	
Commtopo	
Control	CENTRALIZED
Assign	STATIC Memory:TBE
Synch	UNIVERSAL
Max_cpu	"up to 2.5M PEs and interconnections"
Cpu_size	32 bit processing (20Mflops)
Perform	
Market	
Software1	
Software2	
Comment1	use with IBM PC-AT or 80386;"Neurosoft" software; treats neural
Comment2	network as callable "C routines"; 2 or 10MB memory

Name	Armstrong Multicomputer
Company	Brown University
Stream	MIMD
Commtech()	point-to-point (programmable IO interconnect)
Commtopo	reconfigurable (mesh,tree,etc.)
Control	DECENTRALIZED
Assign	TBD Memory: PRIVATE
Synch	CONDITIONAL
Max_cpu	100 (current 68)
Cpu_size	32 (Motorola 68010; National. Semiconductor 32081 floating point)
Perform	(.5MIP individual PEs)
Market	
Software1	C with send/receive extensions for message-passing
Software2	
Comment1	data from [Rayfield and Silverman 1988]
Comment2	

Name	ASPRO (Associative Processor)
Company	Loral Systems Group (formerly Goodyear Aerospace)
Stream	SIMD
Commtech()	
Commtopo	
Control	CENTRALIZED
Assign	TBD Memory:TBE
Synch	UNIVERSAL
Max_cpu	1792 (in application, but can be more)
Cpu_size	16 (indndnt CPU,embd ctrl prcssr); (32bit 68020)
Perform	40Mops [Loral telephone conversation]
Market	NAVY/Grumman E-2C AEW airrft;air/ship track/survll/C3I
Software1	VAX/VMS & UNIX OS;Fortran, OPS-83 (expert system tool)
Software2	and ASPRO assembler
Comment1	ASPRO=smaller,militarized STARAN
Comment2	

Name	Boltzmann Machine
Company	Company
Stream	
Commtech()	
Commtopo	
Control	CENTRALIZED
Assign	STATIC Memory:TBE
Synch	UNIVERSAL
Max_cpu	
Cpu_size	
Perform	
Market	
Software1	
Software2	
Comment1	This has become a 'generic' name for a class of neural
Comment2	network machines

Name	BSP (Burroughs Scientific Processor)
Company	Burroughs Corp.
Stream	SIMD
Commtech()	(see below)
Commtopo	crossbar network (PE-to-Memory)
Control	CENTRALIZED
Assign	STATIC Memory: SHARED
Synch	UNIVERSAL
Max_cpu	16 PEs
Cpu_size	48 (instruction word length)
Perform	50MFLOPS [Miller 1987]
Market	Weather, nuclear energy, seismic, structural anal,ysis economic simulation
Software1	
Software2	
Comment1	Vector machine, pipelined, array organization
Comment2	"horizontal micrrcode" organization [Schwartz 1983]

Name	Butterfly Parallel Processor
Company	Bolt, Beranek and Newman
Stream	MIMD
Commtech()	asynchronous packet-switching MIN
Commtopo	butterfly
Control	DECENTRALIZED
Assign	DYNAMIC Memory: SHARED
Synch	CONDITIONAL
Max_cpu	256
Cpu_size	32 (M68020/81)
Perform	
Market	One of 3 DARPA SDI machines
Software1	Chrysalis semi-Unix OS; C & FORTRAN with parallel extensions
Software2	LISP under development [Dongarra 1987]
Comment1	Mem = when booted physically separate memory chips
Comment2	Configured into virtual memory; all Processors can access all Memories

Name	CDC Star-100
Company	Control Data Corporation
Stream	SIMD
Commtech()	NA uniprocessor
Commtopo	NA uniprocessor
Control	CENTRALIZED
Assign	TBD Memory:TBE
Synch	TBS
Max_cpu	1
Cpu_size	64 (bit operands)
Perform	40 MFLOPS [Hwang 1984]
Market	hydrodynamics, fluid dynamics (Navier-Stokes equations)
Software1	
Software2	
Comment1	Memory-to-Memory; 2 pipes; 40Mflops
Comment2	

Name	CEDAR
Company	University of Illinois
Stream	MIMD
Commtech()	() packet-switch MIN (global);circ-switch crossbar (local)
Commtopo	omega (gbl); shuffle net with 8x8 crossbar (local cluster)
Control	DECENTRALIZED
Assign	DYNAMIC Memory: BOTH
Synch	CONDITIONAL
Max_cpu	32 (4 clusters x 8) as of '87
Cpu_size	64 (from Alliant FX/8 clusters)
Perform	
Market	general
Software1	UNIX-like Xylem OS; Cedar FORTRAN compiler is superset
Software2	of FORTRAN 8x; have FORTRAN77 to Cedar FORTRAN translator
Comment1	Alliant FX/8 clusters with global shared memory
Comment2	

Name	Celerity 6000 [ ]
Company	Celerity
Stream	SIMD
Commtech()	
Commtopo	
Control	UNKNOWN
Assign	TBD Memory:TBE
Synch	TBS
Max_cpu	4 processors (+ vector coprocessor)
Cpu_size	unknown
Perform	
Market	
Softwar1	automatic vectorization of FORTRAN77 code
Softwar2	
Comment1	vector coprocessor has 8 '1024-element' vector register
Comment2	

Name	CHiP (Configurable Highly Parallel multicomputer)
Company	Purdue University and Washington University
Stream	MIMD
Commtech()	asynchronous programmable circ-switch MIN (Processor-to-Processor)
Commtopo	programmable "switch lattice" mesh can be = tree,torus,etc.
Control	CENTRALIZED
Assign	STATIC Memory: PRIVATE
Synch	UNIVERSAL
Max_cpu	64K (256-64K) prototype had 64
Cpu_size	16 planned (prototype was 8-bits)
Perform	
Market	"Pringle" version solved systems of linear equations
Softwar1	
Softwar2	
Comment1	Gannon-can mimick systolic array; often did in tests
Comment2	Snyder-systolic examples mislead;is reconfigurable topology MIMD

Name	Cm* Machine
Company	Carnegie-Mellon University
Stream	MIMD
Commtech()	() packet-switching bus (P-M)
Commtopo	Intracluser-linear,MAP-bus;interclus-misc,eg,Star
Control	DECENTRALIZED
Assign	DYNAMIC Memory: SHARED
Synch	CONDITIONAL
Max_cpu	50
Cpu_size	16 (DEC LSI-11)
Perform	
Market	Simulation, testbed
Softwar1	
Softwar2	
Comment1	Differs from C.mmp, which used crossbar
Comment2	

Name	CMOS VLSI Neural Network[C]
Company	A T & T Bell Labs
Stream	
Commtech()	VLSI analog
Commtopo	mesh (54x54) programmable interconnections
Control	CENTRALIZED
Assign	STATIC Memory:TBE
Synch	UNIVERSAL
Max_cpu	54 (amplifier pairs serves as PEs)
Cpu_size	54 (effective vector length)
Perform	
Market	(experimental) machine vision; character recognition
Softwar1	
Softwar2	
Comment1	mini front-end; analog IN but digital registers; can
Comment2	organize PEs as vector or 'label' (output match) units

Name	Connection Machine
Company	Thinking Machines Corp.
Stream	SIMD-HillisP.61
Commtech()	synchronous packet-switching MIN
Commtopo	(a) 4x4 meshes (b) connected by hypercube network
Control	CENTRALIZED
Assign	HYBRID Memory: PRIVATE
Synch	UNIVERSAL
Max_cpu	(a) 65,536 PEs (CM-2 model); (b) 16,384 (CM-1)
Cpu_size	1-bit PEs
Perform	1000Mips (expected) [Dongarra 1987]
Market	Image processing, simulation, FFTs, AI (1 of 3 DARPA SDI machine
Softwar1	CM-C (C language); CM-Lisp; REL-2 assembly language
Softwar2	
Comment1	synchronization=microcontroller;Hillis p. 20
Comment2	hybrid assignment

Name	Convex C-1 XL/XP
Company	Convex Computer Corp.
Stream	MIMD
Commtech()	80Mbit/sec fiber optic coaxial cable
Commtopo	
Control	DECENTRALIZED
Assign	TBD Memory:TBE
Synch	CONDITIONAL
Max_cpu	240
Cpu_size	64
Perform	40MFLOPS (32bit);20MFLOPS (64bit); LINPACK-> 3-4 MFLOPS
Market	Aerospace,signal and image processing, seismic, simul ation
Softwar1	vectorizing FORTRAN and C compilers that both accept VAX
Softwar2	VMS FORTRAN statements
Comment1	Vector register-to-register architecture; pipelined
Comment2	Functional units.

Name	Computing Surface
Company	Meiko (Great Britain)
Stream	MIMD/SIMD
Commtech()	asynchronous circuit-switching MIN
Commtopo	reconfig. (hypercube/ring)
Control	DECENTRALIZED
Assign	DYNAMIC Memory: PRIVATE
Synch	CONDITIONAL
Max_cpu	512 (installed); 1024 in progress
Cpu_size	32-bit
Perform	1.2 megaflops per PE (IEEE multiply)
Market	graphics, image processing, simulation,numeric analysis
Softwar1	FORTRAN, C, Pascal,
Softwar2	Occam II for high parallel efficiency
Comment1	TRANSPUTER is basic PE building block
Comment2	

Name	COSMIC CUBE
Company	California Institute of Technology
Stream	MIMD
Commtech()	asynchronous message-passing point-point (P-P)
Commtopo	hypercube (6-cube)
Control	DECENTRALIZED
Assign	HYBRID Memory: PRIVATE
Synch	CONDITIONAL
Max_cpu	64
Cpu_size	16
Perform	
Market	astrophysics,quantum chemistry,fluid and structural mechanics
Softwar1	
Softwar2	
Comment1	direct message-passing system
Comment2	point-to-point communications channels

Name	Cray X-MP/4
Company	Cray Research, Inc.
Stream	MIMD
Commtech()	DMA
Commtopo	Star (all PEs have parallel DMA to shared central memory)
Control	CENTRALIZED
Assign	DYNAMIC Memory: SHARED
Synch	CONDITIONAL
Max_cpu	4
Cpu_size	64
Perform	940Mflops (235/PE x 4) [Dongarra 1987]
Market	Scientific and engineering
Softwar1	Vectorizing compilers for Cray FORTRAN (CFT), CFT77, C
Softwar2	Pascal; OS = COS, UNICOS (Unix version), CTSS
Comment1	Multitasking or multiprogramming modes
Comment2	Pipelined vector architecture; multiple functional units

Name	Cray-1
Company	Cray Research Inc.
Stream	SIMD
Commtech()	NA uniprocessor
Commtopo	NA uniprocessor
Control	CENTRALIZED
Assign	TBD Memory: PRIVATE --
Synch	UNIVERSAL
Max_cpu	1 scalar master; 12 "unifunction" pipes
Cpu_size	32 (and 16-bit) instruction length
Perform	160Mflops [Dongarra 1987]
Market	Scientific and engineering
Softwar1	automatic vectorization of Cray FORTRAN (CFT) code
Softwar2	
Comment1	pipelined SIMD; Register-to-Register machine
Comment2	

Name	Cray-1
Company	Cray Research Inc.
Stream	SIMD
Commtech()	NA uniprocessor
Commtopo	NA uniprocessor
Control	CENTRALIZED
Assign	TBD Memory: PRIVATE
Synch	UNIVERSAL
Max_cpu	1 scalar master; 12 "unifunction" pipes
Cpu_size	32 (and 16-bit) instruction length
Perform	160Mflops [Dongarra 1987]
Market	Scientific and engineering
Softwar1	automatic vectorization of Cray FORTRAN (CFT) code
Softwar2	
Comment1	pipelined SIMD; Register-to-Register machine
Comment2	

Name	Cyberplus
Company	Control Data Corp.
Stream	MIMD
Commtech()	synchronous packet-switched decentralized MIN
Commtopo	multiring packet switch; 16bit rings:host-to-processor,processor-to-proc
Control	DECENTRALIZED
Assign	STATIC Memory: PRIVATE
Synch	UNIVERSAL
Max_cpu	64
Cpu_size	64
Perform	(100/PE x 256 max) [Dongarra 1987]
Market	numerical analysis, signal processing
Softwar1	assembler, FORTRAN compiler
Softwar2	
Comment1	PEs - 2 internal crossbars unite multiple functional
Comment2	units within PEs

Name	DADO2
Company	Columbia University
Stream	SIMD/MIMD
Commtech()	"specialized IO switch" tree-MIN
Commtopo	complete binary tree
Control	DECENTRALIZED
Assign	STATIC Memory: PRIVATE
Synch	UNIVERSAL
Max_cpu	1023
Cpu_size	8
Perform	
Market	AI prod sys;speech recog;sonar;digital signal proc
Softwar1	Parallel C, Lisp, PL/M [Stolfo 1987]
Softwar2	
Comment1	32-boards;under 2 cubic ft; possible VHSIC vers from
Comment2	Raytheon, Martin M.

Name	DAP (Distributed Array Processor)
Company	International Computer Limited (ICL-Eng)
Stream	SIMD
Commtech()	(synchronous)
Commtopo	mesh, various config...32x32...256x256
Control	CENTRALIZED
Assign	STATIC Memory:SHARED
Synch	UNIVERSAL
Max_cpu	4096 (64x64 grid); each bit PE has 4096 bits of memory
Cpu_size	1-bit PEs
Perform	
Market	Numerical, Monte Carlo simulation; image processing
Softwar1	Unix host has cross-compilers and run-time debugger;
Softwar2	Enhanced Fortran compiler has most FORTRAN8x array extensions
Comment1	similar to MPP bit-plane organization
Comment2	bit-serial,word-parallel operations;DAP-3 ret to 32x32 PE

Name	Data Driven Machine 1
Company	University Utah (A.L. Davis)
Stream	
Commtech()	1x8 switch at each (tree node) PE
Commtopo	8-ary tree
Control	UNKNOWN
Assign	DYNAMIC Memory : PRIVATE
Synch	CONDITIONAL
Max_cpu	
Cpu_size	
Perform	tokens aren't tagged; token storage and enabling counter
Market	
Softwar1	
Softwar2	data-driven,expression-maipul-organization
Comment1	note: according to Dr. Srini (UCB) DDM2 project terminated
Comment2	

Name	ELI (Enormously Longword Instruction)
Company	Yale University (Joseph Fisher)
Stream	
Commtech()	() clusterbus
Commtopo	ring (nearest neighbor and some 'removed' communication)
Control	UNKNOWN
Assign	TBD Memory : PRIVATE
Synch	TBS
Max_cpu	16 (in 'cluster' ring)
Cpu_size	500+ (this was project's emphasis)
Perform	
Market	
Softwar1	
Softwar2	
Comment1	This project was terminated in 1984
Comment2	Apparently no machine was ever built at Yale

Name	ELXSI System 6400
Company	ELXSI (sub of Trilogy, Ltd.)
Stream	MIMD
Commtech()	Bus
Commtopo	linear
Control	UNKNOWN
Assign	TBD Memory :SHARED
Synch	TBS
Max_cpu	12 (up to 16?)
Cpu_size	64
Perform	
Market	
Softwar1	EMBOS message-based OS and Elxsi version of Unix
Softwar2	FORTRAN77, Pascal, COBOL74, C, MAINSAIL
Comment1	
Comment2	

Name	Encore Multimax
Company	Encore Computer Corp.
Stream	MIMD
Commtech()	synchronous shared bus (P-M)
Commtopo	linear (wide "Nanobus")
Control	CENTRALIZED
Assign	DYNAMIC Memory :SHARED
Synch	CONDITIONAL
Max_cpu	20
Cpu_size	32 (National Semiconductor 32032 or 32332 w/ 32081flpt)
Perform	15Mips (quoted) [Dongarra 1987]
Market	general-purpose [Encore data sheet 1988]
Softwar1	Unix 4.2 with C, Pascal, FORTRAN
Softwar2	
Comment1	Emphasis: fast bus, shared memory with private PE cache
Comment2	Medium/coarse grain parallelism

Name	ETA-10
Company	ETA Systems, Inc. (Control Data subsidiary)
Stream	MIMD/(M)SIMD
Commtech()	DMA between CPUs and Shared Mem
Commtopo	each CPU direct to Shared Memory
Control	CENTRALIZED
Assign	DYNAMIC Memory :SHARED
Synch	CONDITIONAL
Max_cpu	8 (+18 I/O processors and coordinating service processor)
Cpu_size	64
Perform	10,000Mflops (1250/PE x 8) [Dongarra 1987]
Market	Scientific and engineering
Softwar1	Virtual mem OS; UNIX compatible; Auto-Vectorizing FORTRAN
Softwar2	debugger, performance analyzer, code maintenance tools
Comment1	4MB local memory + 256MB shared memory
Comment2	cpu = scalar unit + double-pipelined vector

Name	FACOM Vector Processing System (VP-200)
Company	Fujitsu Ltd.
Stream	SIMD
Commtech()	NA - uniprocessor
Commtopo	NA - uniprocessor
Control	CENTRALIZED
Assign	STATIC Memory : PRIVATE
Synch	UNIVERSAL
Max_cpu	1 scalar, 1 mask, 6 vector pipes
Cpu_size	64 (flptops)
Perform	1142 (400 model) [Dongarra 1987]
Market	numerical (VLSI design, oil and nuclear simulation)
Softwar1	auto vectorizing FORTRAN; interactive debugger and vectorizer
Softwar2	perf. analyzer & scientific library (223 routines)
Comment1	pipelined; multiple funct'al units; register-to-register machine
Comment2	decoding and scalar operations unit

Name	Fifth Generation Computer System (FGCS)
Company	University of Tokyo
Stream	
Commtech()	
Commtopo	
Control	DECENTRALIZED
Assign	TBD Memory :BOTH
Synch	TBS
Max_cpu	approximately 1000 PEs (tological inferences)
Cpu_size	
Perform	
Market	
Softwar1	
Softwar2	
Comment1	not an architecture per se; a concept emphasizing DB-architectures,
Comment2	speech encoding; direct PROLOG execution;AI Production System

Name	FLEX/32 Multicomputer
Company	FLexible Corporation
Stream	MIMD
Commtech()	(VME) Bus
Commtopo	
Control	DECENTRALIZED
Assign	TBD Memory :SHARED
Syncn	CONDITIONAL
Max_cpu	"Claimed limit" = 20480
Cpu_size	32-bit
Perform	1 Mip/PE (NS 32032); 1 Mflop/PE with floating point accelerator
Market	
Softwar1	UNIX System V on each PE with concurrency extensions
Softwar2	FLEX's MMOS real-time OS; FORTRAN77, Ratfor, C
Comment1	Flexible configurations of local and common memory
Comment2	

Name	Galaxy (YH-1)
Company	People's Republic of China
Stream	SIMD
Commtech()	NA - uniprocessor
Commtopo	NA - uniprocessor
Control	UNKNOWN
Assign	STATIC Memory : PRIVATE
Synch	TBS
Max_cpu	1
Cpu_size	
Perform	120Mflops [Hwang 1984]
Market	
Softwar1	
Softwar2	
Comment1	register-to-register vector architecture
Comment2	

Name	GF11
Company	IBM
Stream	SIMD
Commtech()	Memphis switch (P-M)
Commtopo	programmable Benes net.; hypercubic lattice for QCD
Control	CENTRALIZED
Assign	STATIC Memory :SHARED
Synch	UNIVERSAL
Max_cpu	566 Processor boards (each = 4 floating point units, 2 multipliers)
Cpu_size	32-bit floating point chips
Perform	11.4 Gflops
Market	quantum chromodynamics (QCD) calculations
Softwar1	microcode on PEs, C augmented with special procedures on control
Softwar2	cpu; Pascal on IBM 3090 host for algorithm expression
Comment1	3 stage memory per board; 256word register file,16K static
Comment2	RAM, 512K dynamic RAM

Name	HEP
Company	Denelcor
Stream	MIMD
Commtech()	Synchronous pipelined packet-switching MIN (P-M)
Commtopo	Reconfigurable graph
Control	DECENTRALIZED
Assign	STATIC Memory :SHARED
Synch	CONDITIONAL
Max_cpu	16 (variable)
Cpu_size	64
Perform	160Mflops (10/PE x 16) [Dongarra 1987]
Market	General-purpose and scientific
Softwar1	Unix III, linear algebra kernels; FORTRAN77, C, Pascal
Softwar2	
Comment1	Pipelining and multiple functional units, parallelism
Comment2	At the process level; Denelcor no longer exists

II-31

Name	Hitachi S-810
Company	Hitachi
Stream	SIMD
Commtech()	NA - uniprocessor
Commtopo	NA
Control	CENTRALIZED
Assign	STATIC Memory : PRIVATE
Synch	UNIVERSAL
Max_cpu	1
Cpu_size	
Perform	840Mflops [Dongarra 1987]; 500Mflops [Hwang 1984]
Market	Scientific and engineering
Softwar1	
Softwar2	
Comment1	register-to-register pipelined architecture
Comment2	

Name	Illiac IV
Company	Burroughs
Stream	SIMD
Commtech()	(synch) bus (P-P)
Commtopo	8x8 mesh (equiv "nearest neighbor")
Control	CENTRALIZED
Assign	STATIC Memory : PRIVATE
Synch	UNIVERSAL
Max_cpu	64
Cpu_size	64 (can parition as 2-32bit,etc)
Perform	
Market	Scientific (partial differential equations)
Software1	
Software2	
Comment1	Non-pipelined [Jordan 1983]; lockstep operation
Comment2	

Name	iPSC (Intel Personal SuperComputer)
Company	Intel
Stream	SIMD
Commtech()	Asynchronous MIN
Commtopo	Hypercube
Control	UNKNOWN
Assign	TBD Memory : PRIVATE
Synch	CONDITIONAL
Max_cpu	128
Cpu_size	16
Perform	1280Mflops (short precis.;64-node) [Dongarra 1987]
Market	Scientific
Software1	Microsoft Xenix 3.0; FORTRAN, C, LISP, ASM286,
Software2	FCP (Flat Concurrent Prolog); Debugger
Comment1	(node/PEs based on 80286 chip)
Comment2	Private memory, message-passing system

Name	Matrix-1
Company	Saxpy Computer Corp.
Stream	SIMD
Commtech()	synch circuit-switched "partial crossbar"
Commtopo	3 data paths: systolic,SIMD-broadcast, local memory-Mtx-..face
Control	CENTRALIZED
Assign	STATIC Memory :BOTH
Synch	UNIVERSAL
Max_cpu	32 (8,16,24)
Cpu_size	32
Perform	1000Mflops [Foulser 1987]
Market	signal-processing, matrix operations
Software1	"VMS" FORTRAN 77, Pascal, Ada, C [Dongarra 1987]
Software2	Matrix math routine libraries
Comment1	programmable systolic architecture
Comment2	

Name	MIT Data-Flow Computer
Company	MIT (Dennis - 1979)
Stream	MIMD
Commtech()	() packet-switching BUS
Commtopo	
Control	UNKNOWN
Assign	TBD Memory :SHARED
Synch	CONDITIONAL
Max_cpu	
Cpu_size	32-bits (in Mandala project [Srini 1986])
Perform	
Market	
Software1	Ackerman's single-assignment "VAL" language influenced
Software2	project (see [Treleaven 1982])
Comment1	Not built, but concepts used by others [Srini 1986]
Comment2	'Static' token storage architecture

Name	MPP (Massively Parallel Processor)
Company	Loral Systems Group
Stream	SIMD
Commtech()	Bi-directinal data bus (PE groups on VLSI chip)
Commtopo	128x128 nearest neighbor mesh (programmable wrap)
Control	CENTRALIZED
Assign	STATIC Memory :SHARED
Synch	UNIVERSAL
Max_cpu	16384 1b-PE/"plane" and aray control unit for scalar arit
Cpu_size	1-bit (for 16,384 array PEs); 64-bit for control unit
Perform	400Mflops [Dongarra 1987]
Market	Satellite imagery
Software1	Parallel Pascal
Software2	
Comment1	
Comment2	

Name	NCUBE/10
Company	NCUBE (Beaverton, Ore)
Stream	MIMD
Commtech()	
Commtopo	hypercube
Control	DECENTRALIZED
Assign	TBD Memory : PRIVATE
Synch	CONDITIONAL
Max_cpu	1024
Cpu_size	32
Perform	0.3-0.5 megaflops per node [Wiley 87]
Market	
Software1	
Software2	
Comment1	special single chip 32-bit CPU with 11 bidirectional comm
Comment2	channels and memory controller [Wiley 1987]

Name	NEC SX-2
Company	NEC
Stream	SIMD
Commtech()	NA - uniprocessor
Commtopo	NA - uniprocessor
Control	CENTRALIZED
Assign	STATIC Memory : PRIVATE
Synch	UNIVERSAL
Max_cpu	1 scalar, 1 mask unit, 16 parallel units/4 vector pipes
Cpu_size	
Perform	1300Mflops [Dongarra 1987], [Hwang 1984]
Market	Scientific and engineering
Software1	Auto vectorizing FORTRAN77; vectorizing and analyzer tools;
Software2	ALGOL,PL/1,BASIC,Pascal,LISP,C,PROLOG,COBOL
Comment1	register-to-register, pipelined architecture
Comment2	

Name	NETL
Company	Carnegie-Mellon University (Dr. Scott E. Fahlman)
Stream	
Commtech()	
Commtopo	
Control	CENTRALIZED
Assign	TBD Memory : PRIVATE
Synch	UNIVERSAL
Max_cpu	
Cpu_size	!!! 4/27 phone conversation with Fahlman - project terminated
Perform	
Market	store and access "assertions"
Software1	
Software2	
Comment1	AI 'connectionist' architecture to implement semantic
Comment2	networks (i.e., semantic relation graphs)

Name	Neural Phonetic Typewriter
Company	Helsinki University of Technology (Dr. Teuvo Kohonen)
Stream	
Commtech()	bus
Commtopo	
Control	CENTRALIZED
Assign	STATIC Memory : TBE
Synch	UNIVERSAL
Max_cpu	4=PC host + 2 std. signal processing chips + 80186
Cpu_size	(see above)
Perform	
Market	phonetic transcriptions of Finnish and Japanese
Software1	
Software2	
Comment1	coprocessor-board=80186(control,routing),TMS32010(2 for FFT)
Comment2	virtual neurons, speech learning templates compute on PC

Name	NON-VON(1/3)
Company	Columbia University
Stream	MIMD/mulSIMD
Commtech()	Log-state interconnect (P-P)
Commtopo	Undecided (omega,butterfly,banyan) family
Control	DECENTRALIZED
Assign	TBD Memory :SHARED
Synch	CONDITIONAL
Max_cpu	>16K
Cpu_size	8-bit SmallPEs and ?(microprocessor) LargePEs
Perform	
Market	General-purpose; scientific; DB; image/signal processing
Software1	
Software2	
Comment1	Tree-structure; Large and small PEs; smart disk control system
Comment2	N-V3 appears cancelled; Shaw and Hillyer left Columbia

Name	PASM (Partitionable SIMD/MIMD)
Company	Purdue University
Stream	SIMD/MIMD
Commtech()	Asynchronous circuit-switch MIN (3nets:instrshare, mem,cry IO)
Commtopo	"generalized cube"
Control	DECENTRALIZED
Assign	DYNAMIC Memory : PRIVATE
Synch	UNIVERSAL
Max_cpu	1024 (16 in prototype)
Cpu_size	16 (M68010)
Perform	
Market	Image understand; speech recog and biomedical signal proc
Software1	"PASMOS" O.S. is distributed among PEs
Software2	
Comment1	Memory controllers each synchronize PEs with broadcast,
Comment2	Universal Memory synchornization

Name	RP3 (Research Parallel Processor Project)
Company	IBM
Stream	MIMD
Commtech()	circuit-switching (+ packet-switch queueing) MIN (P-M)
Commtopo	Lawrie's Omega and SW Banyan (2 networks)
Control	UNKNOWN
Assign	DYNAMIC Memory :SHARED
Synch	CONDITIONAL
Max_cpu	512
Cpu_size	32-bits
Perform	1 GIPS
Market	scientific, VLSI design automation
Software1	BSD 4.2 Unix as O.S.; C and FORTRAN
Software2	
Comment1	Shared memory and private mem.ory message-passing OR user mix.
Comment2	Dynamic memory global/local allocation; mark data as cacheable

Name	SPUR (Symbolic Processing Using RISCs)
Company	University California, Berkeley
Stream	MIMD
Commtech()	bus (TI "NuBus")
Commtopo	(PEs and shared memory linked by a single bus)
Control	DECENTRALIZED
Assign	TBD Memory :SHARED
Synch	CONDITIONAL
Max_cpu	6-12 (general-purpose PEs with LISP and floating point. support)
Cpu_size	40 (32 + 8 for LISP tags in 64b words) 32 for Non-LISP
Perform	
Market	workstation; parallel LISP
Software1	Common LISP
Software2	
Comment1	128K bytes of cache per PE, using virtual addresses.
Comment2	38-bit global virtual addresses in 256G-byte virtual space

Name	STARAN
Company	LORAL (original builder = Goodyear Aerospace)
Stream	SIMD
Commtech()	FLIP network (within each Associative module)
Commtopo	
Control	UNKNOWN
Assign	TBD Memory :SHARED
Synch	TBS
Max_cpu	8192: 32 associative arry modules with 256 simpl ePEs each
Cpu_size	1 (256 PEs within each associative module)
Perform	approx. 80Mops [Loral telephone conversation]
Market	Cartography; image/signal proc;stereophotogrammetry
Software1	
Software2	
Comment1	Used for cartography at Defense Mapping Agency
Comment2	

Name	Systolic Adaptive Beamformer
Company	ESL, Inc.
Stream	MIMD
Commtech()	direct point-to-point
Commtopo	
Control	CENTRALIZED
Assign	STATIC Memory : TBE
Synch	UNIVERSAL
Max_cpu	
Cpu_size	32-bit VLSI chip - floating point add/multiply for complex values
Perform	350MFLOP
Market	acoustic signal processing (esp. sonar)
Software1	
Software2	
Comment1	Uses custom VLSI chips; input from 100 sensor channels
Comment2	Results output to VAX-11/750; [Kandle 1987]

Name	Systolic/Cellular System
Company	Hughes Research Laboratories
Stream	MIMD/SIMD
Commtech()	
Commtopo	16 x 16 mesh
Control	CENTRALIZED
Assign	STATIC Memory : TBE
Synch	UNIVERSAL
Max_cpu	256
Cpu_size	32-bit
Perform	450MOPS
Market	signal processing (Faddeeva and Luk algorithms)
Software1	
Software2	
Comment1	Operates in 'cellular' or systolic 'modes' [Nash 1987]
Comment2	Dual-port array memory and PE memory

Name	T-ASP (Teamed Architecture Signal Processor)
Company	Motorola (Canada)
Stream	SIMD
Commtech()	
Commtopo	cube
Control	DECENTRALIZED
Assign	TBD Memory :SHARED
Synch	TBS
Max_cpu	8 "fully pipelined vector processors"
Cpu_size	40 bits (for complex #s)
Perform	320Mflops [Lang, et. al.1988]
Market	passive and active sonar; satellite data processing
Software1	T-ASP OS (TOS) supports real-time mutiluser & multitask
Software2	MLP signal-processing langanguage.; debuggers, sig-proc lib, mem ed.
Comment1	complex-number-format; 2 memory caches and interleaved memory
Comment2	3 controllers - arithmetic, transfer and communications

Name	Tagged Token Dataflow Machine
Company	MIT (Arvind Machine)
Stream	MIMD
Commtech()	Shared bus (M-M); switches emulated misc INs (P-P)
Commtopo	Emulates misc. INs and topologies
Control	UNKNOWN
Assign	DYNAMIC Memory : PRIVATE
Synch	TBS
Max_cpu	32-Symbolics Corp. emula.;256-Electrotechnical Lab(Jap)
Cpu_size	32-bits
Perform	
Market	
Software1	high-level "Id" language [Srini 1986, pp. 78-9)]
Software2	
Comment1	Operand-driven; dataflow machine
Comment2	Packet Communications organization; token-matching

Name	TI-ASC (Advanced Scientific Computer)
Company	Texas Instruments
Stream	SIMD
Commtech()	
Commtopo	
Control	UNKNOWN
Assign	TBD Memory : TBE
Synch	TBS
Max_cpu	1
Cpu_size	
Perform	40Mflops [Hwang 1984]
Market	seismic, fluid dynamics, defense
Software1	
Software2	
Comment1	pipelined, memory-to-memory architecture
Comment2	

Name	TRAC v1.1 (Texas Reconfigurable Array Computer)
Company	Univ. Texas, Austin
Stream	SIMD/MIMD
Commtech()	() packet(mem) and prog'rammable circuit switching MIN (P-M)
Commtopo	banyan (SW-fanout=3,spread=2,levels=2)
Control	CENTRALIZED
Assign	DYNAMIC Memory :SHARED
Synch	UNIVERSAL
Max_cpu	4
Cpu_size	8 ("byte-sliced" microprocessor in v.1.1)
Perform	
Market	
Software1	
Software2	
Comment1	emph:"inductive" arch for expansion and dynamic programming
Comment2	[Malek conv.]; most mem private; reconfigurable tree/buses

Name	Ultracomputer
Company	New York University
Stream	MIMD
Commtech()	(asyn?) message-switching [VLSI] MIN (P-M)
Commtopo	Omega
Control	DECENTRALIZED
Assign	DYNAMIC Memory :SHARED
Synch	CONDITIONAL
Max_cpu	4096
Cpu_size	16 (M68010 - currently)
Perform	
Market	General-purpose
Software1	FORTRAN, C, Pascal, (LISP Prolog under development)
Software2	
Comment1	Emphasize shared memory with fetch and add primitive for
Comment2	synchronization and coordination

Name	WARP
Company	Carnegie-Mellon University
Stream	MIMD
Commtech()	Synchronous backplane word transfer
Commtopo	Nearest (left and right) neighbor
Control	CENTRALIZED
Assign	STATIC Memory : PRIVATE
Synch	UNIVERSAL
Max_cpu	10
Cpu_size	32 bits
Perform	100Mflops (based on 10Mflops/PE - [Miller 1987])
Market	Computer vision, signal processing, pd-equations
Software1	
Software2	
Comment1	Programmable systolic architecture; 2-way systolic flow
Comment2	DARPA project; INTEL working on single chip version

CHAPTER III: NON-VON NEUMANN APPLICATIONS ANALYSIS

3.1 INTRODUCTION

Computers that embody NvN architectures potentially offer the computational power required to run applications in the problem domains covered in this study. The classification scheme produced in the Subtask 1, NvN Architecture Study, provides the basis for correlating NvN computers and applications. A problem domain is analyzed by looking at the applications functioning under each of the architecture classifications formulated in the first subtask. This analysis shows the extent each architecture class covers in the selected problem domain.

The survey information presents a variety of applications currently running on each architecture. The existence of an application on a computer system can be attributed to one of two factors: 1) the architecture was designed to solve problems in that application domain, or 2) the computer was available, so the application was transported to it. In either case, the efficiency of an application depends upon the mapping of the algorithm to the architecture and the efforts of the application developers.

The principal factor in determining application performance is the selected algorithms for solving the components of a problem. An inappropriate algorithm impedes the potential of a computer more than any other factor in assessing performance. Once an appropriate algorithm is selected, the ability of the application developer to utilize an architecture determines the final performance characteristics. This reports decomposes a problem domain into its major components, identifies known algorithms for solving these components, and assesses the applicability of the algorithms to NvN architectures.

This report summarizes the information gathered over the past few months on the capacity of NvN computer architectures for solving problems. Section 3.2 presents an analysis of BM/C³I applications and the potential use of NvN computers in BM/C³I. Sections 3.3 to 3.7 report on the use of NvN architectures in the problem domains of Artificial Intelligence, Real-time Simulation, Signal Processing, Image Processing and Use in Development, Prototyping, and Test of Hardware and Software; respectively.

The remainder of this section summarizes the five application areas that are analyzed in this report; BM/C³I, image processing, signal processing, artificial intelligence, and real-time simulation.

3.1.1 BM/C³I

Battle Management, Command, Control, Communications, and Intelligence (BM/C³I) Systems are being analysed as a first step in transitioning them onto the next generation of hardware architectures. Existing, operational BM/C³I systems, based on the traditional von Neumann machine architecture, are hard-pressed to cope with the explosion of information that is required by command authorities

in order to successfully manage modern missile-type weapons on battlefields of global, or near-global scope.

The challenge to the Life Cycle Support Agencies which are responsible for providing computer-based BM/C³I systems to the combat units is to determine how to apply the fruits of on-going research and development of the new non-von Neumann machine architectures to existing and near-term planned BM/C3I systems. If there is a single message arising from the many research and development projecting investigating the NvN machines, it is that the BM/C³I systems of the 1990s are going to be complex aggregates of hardware and software organized into networks of federated clusters of perhaps all seven of the NvN architectures, operating in conjunction with machines of the tradition von Neumann type.

3.1.2 Artificial Intelligence

Section 3.3 discusses the application area of Artificial Intelligence (AI). Specifically, it discusses the potential for implementing production systems on NvN machines. A production system is a rule-based program that executes in a cyclic manner (i.e., a set of conditional rules that are evaluated and acted on iteratively). Production systems are comprised of working memory, a set of rules and a program that evaluates the rules based on the current state of working memory. Expert systems are production systems that contain rules derived from human experts.

The fundamental processes that comprise all production systems are initialization, data acquisition, condition evaluation, rule selection, conflict resolution and rule firing. The critical processes that determine performance on von Neumann computers are condition evaluation and rule firing.

Memory access and complex compare instructions limit the performance on conventional systems. For some large real-time applications, data acquisition requires fast data input capability and data preprocessing prior to information being stored in working memory. Response time is critical in real-time applications and inferences must be made in a short time frame.

Memory subsystem speed is likely to be the critical factor in determining the performance of a production system, because matching production preconditions to the current working memories contents consumes the vast majority of compute time. This implies a NvN architecture that balances memory access and conditional evaluation. This aspect has encouraged approaches using both associative memory processors and subtrees of low capacity processors with private memory. Present research suggests that several NvN architecture types can be efficiently exploited for parallel production system execution.

3.1.3 Real-time Simulation

Section 3.4 analyzes the application area of real-time simulation by looking in depth at a specific problem pertinent to Air Defense Initiative (ADI). The ADI Technical Evaluation Facility (TEF)

models the North American Air Defense environment and provides for interaction between simulated real world objects and the simulated effects. This model is complex and contains characteristics found in most real-time simulations. The ADI TEF simulation is comprised of several separate models that are controlled by and communicate through a simulator executive. The TEF executive is a hybrid that combines event stepped simulation with time stepped simulation, thereby providing a centrally controlled discrete event simulation with an underlying selectable time period.

The simulator executive is the key to a successful simulation, and therefore, it should be carefully designed with particular attention given to simulation efficiency and repeatability. For this simulation 10 to 18 minutes is acceptable turnaround time for simulating eleven one-hour time intervals. Examination of the most compute-intensive model revealed processing requirements in excess of 67 MIPS on a von Neumann computer. Moreover, the computer system needs access to over 33 MBytes of real memory and over 2 GBytes of on-line data storage. The large amount of data access and data movement characterizes most simulation applications.

Each of the individual models for object motion, sensor detection or environmental calculations are possible subjects for parallel processing. The calculations performed are identical for all objects of the same category and simultaneous evaluation offers the potential for greatly increased efficiency. For the simulation executive, feasible parallel execution might be the distribution of functions, provided the simulation is repeatable (i.e., executing the simulation with the same input parameters and data result in identical output data). A large grain parallel architecture provides the best choice for the control architecture, with each large processor having the ability to execute fine-grained parallel calculations, such as vector or array processing.

3.1.4 Signal Processng

Signal processing is the application of algorithms to sampled data from single or multiple sensors for the purpose of extracting intelligence from the data and/or improving the quality of intelligence that may be extracted. Signal processing techniques are applied to many types of signals including: telecommunication, radar, video images, acoustic, seismic, and medical instrumentation.

The processing algorithms are applied for a variety of purposes, such as improvement of signal-to noise ratio, speech recognition/speech compression, detection of events, pattern recognition, parameter measurement, and image processing.

The most pervasive problem of signal processing is its computational intensity. In some cases relatively high I/O bandwidths are also required, but computational bandwidth is the predominant problem.

The problem of high data rates from a large number of sensors is exacerbated by the additional requirement for high precision computation when using the more sophisticated processing algo- rithms. Advances in signal processing over the past three decades have brought increasing

complexity of the algorithms, ranging from filtering to spectral analysis to adaptive beamforming. These changes in algorithmic complexity have altered the computational load from a factor of N to a factor of N^2 to a factor of N^3 (where N is the number of data samples to be processed in a given time period). In most signal processing applications, the processing load must be handled in real-time.

A common and significant attribute of most signal processing applications is the use of complex mathematical techniques such as FFT (fast Fourier transform), IIR (infinite impulse response) filtering, FIR (finite impulse response) filtering, and matrix operations. This algorithmic commonality makes it feasible in many instances to select or to design a system architecture that is suitable for multiple signal processing applications.

Non-von Neumann architectures are already in use in most of the signal processing applications where computational bandwidth requirements indicate the need and where cost allows. Numerous pipelined array processors (not to be confused with processor arrays) of the class 1 type have been commercially available as peripherals to mainframe computers, and have been applied to many signal processing applications since the early 1970s.

Adaptive beamforming in radar, sonar, and seismic applications has been performed using rhythmic cellular architectures as well as processor array type architectures. Target tracking applications have also been performed on associative processor architectures. Processor arrays have also been applied to speech and image processing. Various multiple processing element (PE) architectures have been applied to general signal processing, including the application of expert systems technology to signal analysis.

3.1.5 Image Processing

Image processing has been defined in terms of two categories of processing by S.Y. Kung in his book VLSI Array Processors [Kung1988]. The research activities dealing with images are divided into two disciplines: image processing and image analysis. Image processing consists of enhancement, restoration, reconstruction and coding, etc. Image analysis, on the other hand, deals with extraction of lines, curves, and regions in images, classification of objects, texture analysis, analysis of moving objects, and scene analysis. Most image processing tasks are very time consuming. For example, low-level operations, such as filtering or enhancement, typically require on the order of some tens of machine instructions per pixel. A typical image obtained from a LANDSAT earth resources satellite is about 1000 x 1000 pixels/image. This implies a computation requirement of some tens of millions of instructions per image, not including the computation for any substantive higher-level processing. If such simple low-level operations are to be performed at a video rate, say 25 to 30 frames per second, this means a throughput requirement of about a billion instructions per second. In general, most real-time image processing throughput rates outstrip current parallel architectures. Thus image applications processing has long been (and will continue to be) a major driving force in the development of faster and more powerful parallel machines.

3.1.6 General Purpose Use of NvN Machines

Section 3.7 examines the use of NvN architectures for software engineering, from the viewpoint of development, prototyping, and testing of hardware and software and from the perspective of problem domains for which NvN architectures are applicable.

3.2 BATTLE MANAGEMENT/C3I APPLICATIONS

3.2.1 Generic Definition of BM/C3I

At a top-level, Battle Management/Command, Control, Communications and Intelligence (BM/C3I) can be defined as a set of coordinated personnel- and technology-based activities by which a command authority can apply assigned military resources such that the military goals/objectives levied by higher authorities can be achieved in an optimal fashion. A typical set of coordinated activities that make up BM/C^3I is given in the following 22-item list:

- logistics planning for military resource elements
- mission planning for offensive and defensive forces
- maintaining status of forces data (own force and enemy)
- maintaining order of battle data (own force and enemy)
- providing network and point-to-point communications with subordinate units, lateral commands, and higher authorities
- intelligence data gathering
- intelligence data fusion
- disseminating fused intelligence data to command posts
- sensor data acquistition
- multi-sensor correlation
- threat identification, classification, and evaluation
- threat movement tracking
- matching weapon characteristics to target attributes
- weapon assignment
- weapon deployment
- weapon kill evaluation
- maintaining battle area environment information
- maintaining topographical data for battle areas
- large-scale display of topographical data annotated with situation data
- military situation assessment
- support for command decision-making
- preparing and distributing operations orders

During the past decade, the term Battle Management has been used as a modifier of, or sometimes as a replacement for, the older term "Command, Control and Communications". Certainly, a

command authority has always managed, or has attempted to manage, the course of battle. However, with the advent of high-speed aircraft and missile-type weapons that can be launched from ground, air, ocean surface, or submarine platforms the scope of the battlefield has enlarged enormously, and the temporal pace of battle has speeded up considerably. The amount of data and information that a commander must acquire and assimilate in order to make cogent decisions regarding his use of his military resources has increased by several orders of magnitude. The commander can no longer rely on his intuitions borne of his prior experiences in battle; he must rely on a complex network of communications to learn who the enemy is, how he is armed, where he is, and his most likely next attempt to gain the advantage.

The significantly altered scope and pace of modern battle mandates the existence of a powerful battery of computers to store, retrieve, and manipulate the great volumes of data the commander must have if he is to successfully manage the battlefield and emerge as victor from the fray. The term "Battle Management" has come to denote the high-speed, computer-based, large-scale information management that characterizes modern command and control of military resources.

3.2.2 BM/C³I Problems

Effective planning, directing and controlling of offensive and defensive forces in large-scale battles conducted over very broad geographical areas requires ready access to both rapidly-changing information and to information that is basically non-volatile, and an ability to quickly correlate elements of both types of information.

Modern, high-speed, missile-oriented battles can be managed more effectively with modern, high-speed computers that have been made to be intelligent to the extent that they embed a large proportion of the knowledge that has been gained by commanders and their staffs in planning, conducting, and evaluating previous battles.

The problems confronting a BM/C³I command authority are precisely those of making the supporting computer complexes efficient and intelligent.

Some of the information processing problems, particularly those related to the acquisition, analysis, and interpretation of sensor signals, and the processing of image data, as well as those related to augmenting the intelligence of the BM/C³I algorithms, are discussed at some length in other major sections of this report. Table 3-1 relates the typical military tasks given above to generic data processing or computational tasks. The table is presented here as a conceptual aid; it is simpler to relate generic data processing and computational tasks to NvN architectures than it is to relate specifically military tasks to those architectures.

Table 3-1. Matching Military Tasks to Computational Tasks

BM/C3I Military Tasks	Data Processing/Computation Tasks
logistics planning	large database data processing
status of forces	large database data processing
order of battle	large database data processing
communications	managing network traffic and security
intelligence gathering	image and signal processing
intelligence fusion	combined text and image processing
intelligence dissemination	data processing, communications
sensor data acquisition	signal processing
multi-sensor correlation	high-speed computation, pattern recognition, expert systems
threat identification	real-time pattern recognition
threat classification	database, numeric and expert systems
threat evaluation	expert systems
threat tracking	real-time data processing
weapons selection	real-time data processing
weapons assignment	real-time data processing
weapons deployment	real-time data processing, communications
weapons kill evaluation	pattern recognition, communications, high-speed computation
managing topographical data	image data compression/decompression large database processing
displaying topographical data	large-scale graphics
decision-making support	expert system-based forecasting models
mission planning	data processing, expert systems
preparing operations orders	data processing, expert systems
distributing operations orders	data processing, communications

The performance of data processing and/or computational tasks in a BM/C^{3}I arena are problematic primarily because of:

- temporal (real-time or near-real-time) constraints

- the size and scope of the different databases

- the large amount of human knowledge that should be embedded in an intelligent system

- the distribution of functions across geographically dispersed operations and control centers.

Table 3-2 identifies the classes of software that have been used to support BM/C^{3}I tasks within the framework of the traditional von Neumann architecture. With the application of NvN architectures to some of the tasks there emerges the added, new problem of creating system software for the new architectures as well as application software.

Table 3-2. Classes of Software to Support Military Tasks

```
large database data processing
data processing
real-time data processing
high-speed, large-scale computation
network access
network management and control
network security control
image processing
signal processing
pattern recognition
text and image processing
graphical data compression and decompression
large graphical database management
large-scale graphics generation and display
expert systems
message processing
```

The software class titles, given in Table 3-2, are defined as:

(1) Large database data processing—The update, storage, and retrieval of data stored in large-scale media devices. This activity will not usually be performed under real-time or other severe timing constraints. The scope of the databases will be such that a DBMS will be used to interface the BM/C³I algorithms to the data.

(2) Data Processing—The update and retrieval of data from either high-speed or regular storage. This activity will not usually be performed under real-time or other severe timing constraints. Some instances of this activity will make use of a DBMS, while other instances may not require a DBMS because of the small-scale of the data structures holding the data.

(3) Real-Time Data Processing—The update and retrieval of data and/or information in high-speed storage that must be executed in real-time, and will not make use of a DBMS. A typical example of this kind of processing is the detection, classification, evaluation and tracking of threat objects, and the subsequent weapon selection and assignment and deployment.

(4) High-Speed, Large-Scale Computation—The application of complex algorithms to a body of data. This complexity forces extensive use of hardware features such as pipelined vector manipulation. A typical example is multisensor correlation.

(5) Network Access—This class of software provides for the placement of data onto, and the retrieval of data from, the communications medium through which messages will be sent to or received from other nodes in a local area network, or a wide-area network. The key attributes of this software class are robustness and efficiency. Although this class of software is considered to be a support function, its efficiency is vital to the successful performance of military tasks.

(6) Network Management and Control—This class of software assures the on-going reliability of the communications capability that enables easy and rapid inter-nodal communication of battle-sensitive data.

(7) Network Security and Control—this class of software assures that all classifed and/or otherwise sensitive information is handled properly.

(8) Image Processing—The manipulation of images. It is expected that construction, enhancement, and analysis of image data will be performed in an ancillary facility (off-line or detached). Image data can be a considerable support to a command authority in the assimilation of other types of information. Image processing is discussed in the enabling technologies section (Section V) of this report.

(9) Signal Processing—The acquisition, reduction, and analysis of signals from radar or electro-optic sensors. Signal processing is discussed at length in Chapter III of this report.

(10) Pattern Recognition—This permits analysis of signals data and/or image data to precisely identify one or more sensed objects. It enables and supports the making of good command and control decisions.

(11) Text and Image Processing—The merging of text and image data in a single file supports more rapid assimilation of complex intelligence information by commanders and their staff personnel.

(12) Graphical Data Compression and Decompression—This makes use of algorithms that can significantly reduce the amount of storage required for image data. This function is often implemented in hardware.

(13) Large Graphical Database Management—This software manages topographical and other types of image information and supports querying as well as updating.

(14) Large Scale Graphics Generation and Display—This software works in conjunction with that mentioned in the previous item to retrieve graphical and image data and generates displays of the information on a large screen device. This capability supports the rapid assimilation of military situation data by the commander and his staff.

(15) Expert Systems—This is knowledge-based software that embeds the military expertise of commanders and their staff personnel. This type of software assists the commander in making good command and control decisions quickly.

(16) Message Processing—This software takes messages that have been received from other operations and control center nodes and parses the messages and presents them for appropriate display to the center staff personnel.

The Subsection 3.2.4, following, of this BM/C³I section discusses the typical classes of software and identifies those classes for which there is some likelihood that performance can be improved by applying one or more instances of NvN architectures.

3.2.3 Use of NvN Architectures in BM/C³I Applications

Outside certain large-scale test beds, and various Government and industrial research and development laboratories, there is no known usage of NvN architecture machines to existing BM/C³I command computer complexes. There may exist a BM/C³I command computer complex in which NvN machines have been installed to support the mission of that command, but such information has not been available for this task report. RADC is currently managing a contract to define and implement BM/C³I algorithms, using the Center's algorithm testbed.

3.2.3.1 An Object-Oriented Perspective of BM/C3I Systems

For the purposes of discussing the management of data and information, a BM/C³I system can be viewed as a set of six interacting information management objects (Figures 3-1 and 3-2). This object-oriented perspective can be a good conceptual aid in decomposing BM/C³I functions preparatory to allocating them to NvN architectures.
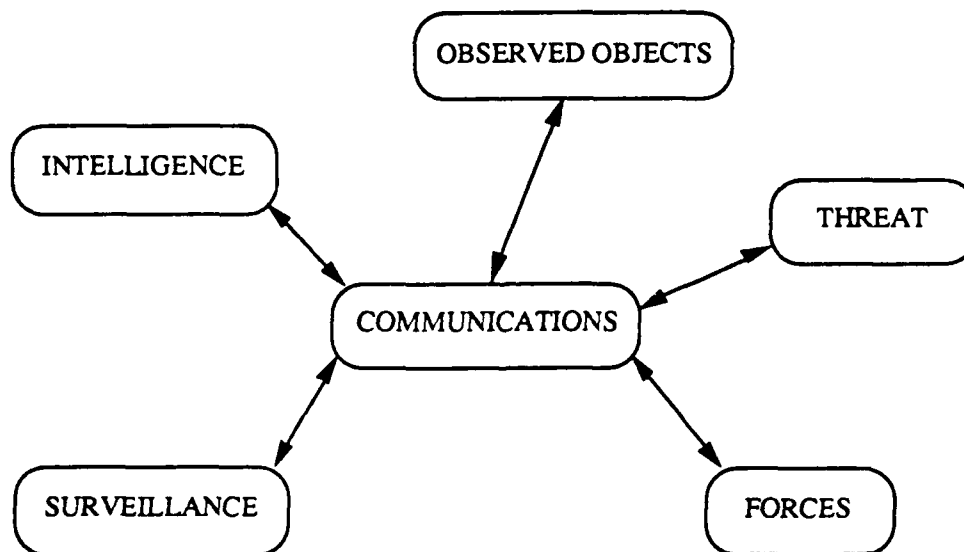
Figure 3-1. The Primary Management Objects of a BM/C3I System

- Surveillance [SURV] Manager—provides location information on observed objects

- Intelligence [INTEL] Manager—provides fused current intelligence to command posts

- Force [FORCES] Manager—manages and controls all information about military resources

- Threat [THREAT] Manager—provides current status of observed hostile objects, provides Weapon Threat Analysis services, and monitors own-force-launched weapons, as well as enemy-launched weapons

- Observed_Objects [OOBJ] Manager—provides descriptive information on observed objects and provides tracking status on observed objects that are moving

- Communications [COMM] Manager—provides for reliable message delivery.

The SURV Manager, through its sensor suites, acquires and distributes location information for objects in its environment.

The INTEL Manager provides fused intelligence information to the various Command Posts in the BM/C3I system. Intelligence information is used to update Operations Plans, Logistic Plans, Enemy Order of Battle Summaries, own-force defended assets status and maintenance of inventories, enemy-force defended assets status, and to support current situation assessment.

The FORCES Manager provides for maintenance and control over Status of Forces data (both own-force and enemy), Environment data, Order-of-Battle data, and for the generation and distribution of Operations Plans and Logistics Plans. Status of Forces data includes current inventory status and readiness status of weapons, accountable equipment, expendable supplies, and personnel.

The THREAT Manager maintains current status information on all tracked objects that have been classified as hostile. It also provides for the generation and dissemination/distribution of weapons directives/orders; for the maintenance of status information on en route weapons launched by Own Forces; and for the generation and distribution of weapon kill evaluation reports.

The OOBJ Manager provides descriptive information for all stationary and moving objects (own-force and enemy) within the purview of the Surveillance Manager. OOBJ also provides tracking data and track histories for all moving objects in the environment of its sensor suite(s).

COMM provides a reliable message delivery service to the other five primary functional objects.

The five top-level objects are made up of sub-objects that provide the top-level objects' characteristic functionality. The sub-objects of the primary objects are identified as follows:
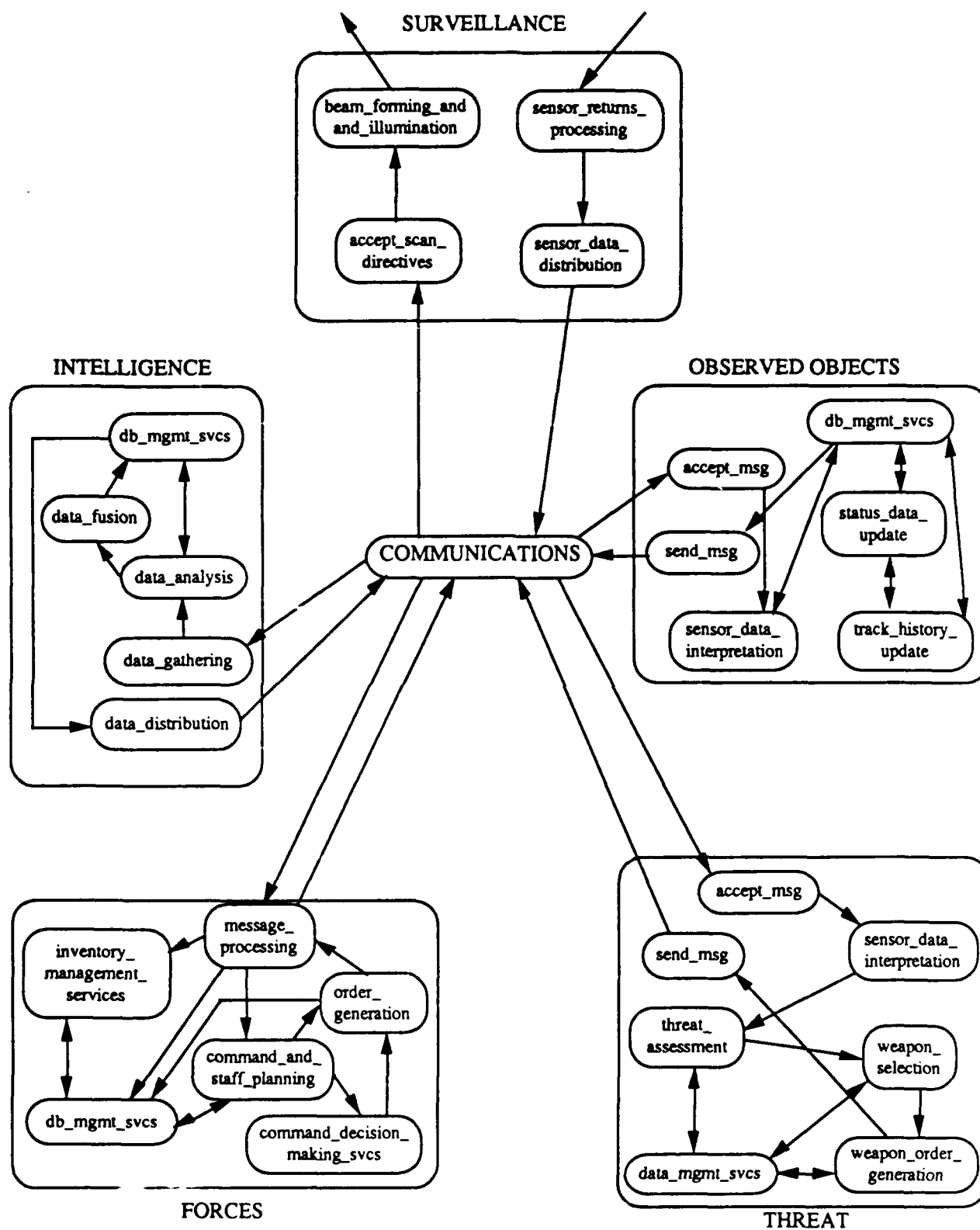
Figure 3-2. The Primary BM/C3I Management Objects and Their Functionality

SURV Manager: accept_scan_directives, beam_forming_and_illumination, returns_processing, data_distribution.

INTEL Manager: data_gathering, data_analysis, data_fusion, data distribution.

FORCES Manager: planning_support, plan_distribution, situation_assessment_support, situation_display, decision_support, inventory_management, environment_data_management, environment_data_presentation, order_of_battle_data_management, topographical_data_management, operations_orders generation, distribution_of_orders/directives.

THREAT Manager: threat_status_data_management, threat_assessment, weapon_selection, weapon_order_generation, weapon_order_distribution, en_route_weapon_status_data_management, interpretation_of_sensor_data, weapon_kill_evaluation, weapon_kill_evaluation_report_generation_and_distribution.

OOBJ Manager: object_description_and_status_data_management, track_histories, track_status_data_management.

1. Specifying Functionality of Primary BM/C³I Objects

In the following paragraphs, the functionality of the subobjects is specified using applicative-language constructs.

The purpose and primary responsibility of the SURV object is the activity: Provide_Sensor_Surveillance_of_Environment. The primary function of SURV is defined in terms of four subsidiary functions.

```
(Provide_Sensor_Surveillence_of_Environment) ==
     (Distribute_Object_Location_Data
         (Process_Illumination_Returns_Data
             (Form_Beams_and_Illuminate_Environment_Sectors
                 (Accept_Sector_Scan_Directive
                     [sector_data_request]$
```

The above construct indicates sequentiality of operations. The first functional operation is the innermost function. For example, in the above construct the Accept_Sector_Scan_Directive operates to accept whatever request data has been prepared by or on behalf of users. The $ closes all unmatched left parentheses.

The INTEL object is responsible for the activity Provide_Intelligence_Information_to_Command_Posts. This primary function is defined in terms of four subsidiary functions, as follows:

```
(Provide_Intelligence_Information_to_Command_Posts)==
    (Distribute_Fused_Intelligence
        (Correlate_and_Fuse_Data
            (Analyse_Gathered_Intelligence_Data
                (Gather_Intelligence_Data [data_sources_output]$
```

The THREAT object is responsible for the activity Evaluate_and_Respond_to_Hostile_Objects. An alternative format for the subsidiary functions is shown in the following construct. In this case the first operation is the first one in the list of functions.

```
(Evaluate_and_Respond_to_Hostile_Objects)==
    (Receive_Track_Data_from_OOBJ);
    (Assess_Threat) ;
        (Maintain_Threat_Status)  |
        (Weapon_Target_Analysis/Weapon_Selection_and_Assignment)  |
    (Maintain_Launched_Weapon_Status);
    (Evaluate_Launched_Weapon_Effect);
    (Generate_Weapon_Kill_Evaluation_Report);
    (Distribute_Kill_Evaluation_Report) $
```

This alternative form illustrates that the two sub-activities of Assess_Threat can be performed in parallel. Each parenthesized function set is followed by a vertical bar; the bar indicates that these activities can be performed in parallel.

The OOBJ is responsible for the activity:

```
(Maintain_Description_and_Location_Data_for_Objects_In_Environment)==
    (Accept_Sensor_Data [SURV_output]);
    (Identify_and_Classify_Objects);
    (Maintain_Object_Description_and_Status_Data);
    (Maintain_Active_Tracks_and_Track_History);
    (Dispatch_Data_to_Other_Objects) $
```

The scope of functionality responsibility of the FORCES object is considerably broader that that of the other objects; consequently, it is necessary to use at least one additional level of abstraction in specifying the top-level functionality. The FORCES object executes the top-level activity Manage_Forces. This top-level activity can be defined in terms of six second-level activities, as shown in the following construct:

```
(Manage_Forces)==
    (Support_Command_and_Staff_Planning_Activities) |
    (Support_Command_Decision_Making) |
    (Manage_Equipment/Supplies/Personnel_Inventories) |
    (Support_Orders/Directives_Generation/Distribution_Activities) |
    (Provide_Database_Management_Services) |
    (Provide_Message_Procesing_Services)$
```

The first function in the above list can be further decomposed, as shown below:

```
(Support_Command_and_Staff_Planning_Activities)==
    (generate_operations_plan(s)) |
    (generate_logistics_plan(s)) |
    (generate_database_queries);
    (receive_database_query_response(s))$
```

This construct shows the two topmost activities can be performed in parallel, but that if either of the topmost function generates a database query, then the receive the results of that query must follow—it can not be done in parallel with the generate query function.

```
(Support_Command_Decision_Making)==
    (assess_new_intelligence_data);
    (situation_assessment);
    (assess_state_of_goal_achievment);
    (graphics_generation);
    (situation_display);
    (decision_support_aids);
    (generate_database_queries);
    (receive_database_query_response(s));
    (generate_orders/directives) |
    (dispatch_orders/directives)$
```

```
(Support_Orders/Directives_Generation/Distribution_Activities)==
    (accept_incoming_message(s));
    (retrieve_order/directive_template(s));
    (complete_order/directive_template(s));
    (generate_message_request(s));
    (dispatch_message_request(s));
    (generate_database_queries);
    (receive_database_query_responses(s))$
```

```
(Provide_Database_Management_Services)==
    [Environment_Database] |
    [Order_of_Battle_Database] |
    [Topographical_Database] |
        (accept_database_transactions/queries);
        (parse_received_transactions/queries);
        (retrieve_requested_data) |
        (write_new_data) |
        (update_existing_data) |
        (delete_existing_data) |
        (generate_database_usage_reports);
        (dispatch_retrieved_data);
        (distribute_reports)$
```

The above construct shows that the functions whose names are indented might be performed on each database named.

```
(Provide_Message_Processing_Services)==
    ((accept_message_requests);
    (generate_message_for_transmission)) |
    ((accept_incoming_messages);
    (parse_incoming_messages)) |
    (prepare_outgoing_messages);
    (transmit_outgoing_messages)$


(Manage_Environment_Database)==
    ((Accept_Sensor_Data) |
    (Accept_Human_Observer_Data));
    (Interpret_Sensor_Data);
    ((Update_Environment_Database) |
    (Accept_Requests_for_Environment_Data));
    (Dispatch_Environment_Data)$


(Manage_Equipment/Supplies/Personnel_Inventories)==
    (Manage_Personnel_Inventories) |
    (Manage_Accountable_Equipment_Inventories) |
    (Manage_Expendable_Supplies_Inventories) |
    (Manage_Weapons_and_Armaments_Inventories)$
```

2. Identification of Major Databases in BM/C3I Systems

The FORCES and INTEL objects have the most complex as well as the most comprehensive database requirements of all the primary functional objects in a BM/C³I command center.

In order to reduce the complexity of the vast set of complicated data interrelationships among the various databases of the FORCES and INTEL functions there should exist a capability to define hierarchies and lattices of complex objects. A complex object can be made up of a set of simple objects, one or more simple objects combined with other complex objects, or a set of other complex objects.

In creating the BM/C³I systems of the 1990s (based on hardware aggregates), the conceptualization and design tasks would be simplified if the database areas associated with the FORCES and INTEL functions were restructured into an object orientation. The higher-level categories of data objects in these functional areas are identified in the following paragraphs.

a. Force Management Databases

The databases necessary to support the management of forces include the following:

Planning Data
> Operations Plans, Logistics Plans [multi-media documents]

Inventory Management and Control
> Personnel Resources

> Accountable Equipment
>> Aircraft, Trucks, Tanks, Weapons Launchers, Sensor Systems, Computers, Radio Receivers, Radio Transmitters, Modems

> Expendable Supplies
>> Foodstuffs, POL Products (Gasoline, Jet Engine Fuel, etc.), Ammunition (Small Arms, Artillery, Bombs, Missiles, Rockets), Paper Products

> Status of Forces
>> Unit Personnel Strength, Readiness Status, Training Status, Equipment Status (Aircraft, Trucks, etc.)

Order of Battle
> Own Forces, Enemy Forces

Topographical and Image Data
> Maps, Photographs, Drawings, Machine-Generated Graphics

Environment Data
   Current Conditions, Predicted Conditions (for entire battle area by grid cell)
   Events Having Environmental Impact (i.e. NUDETS) (for all affected grid cells)

Knowledge Base (Expertise of Personnel)
   Human Knowledge about C2, Comm, Intel, Battle Management, Military Psychology, Planning, Decision-Making, etc. (Much of the Knowledge Base will be made up of symbolic data)

   b. Intelligence Databases

The INTEL databases include the following:

   Enemy Order of Battle (persons, personalities, modii operandi, unit identification, unit strength, unit training/readiness, etc.)

   Status of Enemy Defended Assets

   Status of Own Force Defended Assets

In order to be maximally supportive of the Command and Control of military resources, the various databases and data sets that are incorporated into the INTEL object should be based on multi-media capabilities, such as text combined with photographic images, maps, drawings, and/or video images.

3.2.4   Projected Future Use of NvN Architectures in BM/C3I Applications

3.2.4.1 Large Database Data Processing

BM/C3I operations and control centers use several different large databases. In general, Class VI machines (GPMPE) would be good hosts for large databases since a database could be distributed across multiple storage devices, with each device being controlled by one of the processors. In addition, it should be possible to improve performance of some classes of database searches with NvN Class IV (Associative Array Processor) machines.

3.2.4.2 Data Processing

This type of activity will usually be oriented to accessing data in primary high-speed storage. Some particular processing algorithms might be parallelized; however, it is difficult to make a general statement about the applicability of NvN architecture machines.

3.2.4.3 Real-Time Data Processing

The typical application—identifying, classifying, and evaluating threat objects, and the follow-on task of selecting an appropriate weapon—is a likely candidate for Class VI machines (General Purpose, Multiple-PE Architecture) ([Gottschalk 1987] and [Baillie, Gottschalk and Kolawa 1987]).

3.2.4.4 High-Speed, Large-Scale Computation

A Class I machine (Pipelined Vector Uniprocessor) would most likely be the best fit. The primary processing host in a BM/C³I command node aggregate could well be a Class I machine.

3.2.4.5 Network Access

This is one of the primary support activities for the basic BM/C³I tasks; this function is usually located in a bus interface processor. The set of bus interface processors within a local area network could , in a sense, be viewed as a Class VI.1.1 machine (Bus Connected, General Purpose, Multiple-PE Architecture).

3.2.4.6 Network Management and Control

This is a second element of the communications support capability for the basic BM/C3I tasks. This function is often located in a separate processor rather than in the primary processor.

3.2.4.7 Network Security Control

This is a third major element of the communications support capability. Managing a comprehensive multi-level access control list, could be simplified with a Class IV (Associative Array processor) machine that is attached to a traditional von Neumann (TvN) processor within a processing aggregate.

3.2.4.8 Image Processing

The current state of image processing technology supports the processing of digitized photographs; this capability could be very useful to the intelligence gathering, fusion and dissemination activities that provide fused intelligence information to BM/C3I command authorities. Several of the NvN classes could be applied to such image processing (refer to the enabling technologies section of this report)

3.2.4.9 Signal Processing

The needs of the signal processing community have supplied much of the impetus in developing systolic array and wavefront processors over the past decade. Both types are directly applicable to BM/C³I sensor data acquisition and analysis [Korelsky, et. al. 1988].

3.2.4.10 Pattern Recognition

This is an aspect of computer vision and other image processing activities. It is useful in analyzing sensor data and photographic images to determine the presence or absence of particular objects. This technology is quite applicable to threat identification and threat evaluation. NvN architectures have been rather heavily used in pattern recognition research tasks [Ahuja and Swamy 1984].

3.2.4.11 Text and Image Processing

The previous three paragraphs all apply to the creation of single files containing interspersed textual matter and image data (photographic, topographical maps, line drawings, et cetera). Fused intelligence information will probably be disseminated in such form; this information can be displayed to support command decision-making or can be used to update one or more databases.

3.2.4.12 Graphical Data Compression and Decompression

This technology could be applied to image data to reduce the amount of storage required. There exist both software and hardware implementations of the compression/decompression algorithms. This is an indirect advantage to a BM/C³I command computer complex. This technology could be applied whether the machine holding the image data were attached to a TvN or an NvN machine.

3.2.4.13 Large Graphical Database Management

A general statement cannot be made with respect to the applicability of NvN architectures to this important task. However, it is likely that different databases could each be managed by separate processors of a parallel architecture machine.

3.2.4.14 Large-Scale Graphics Generation and Display

This is a companion task to the previous task. It is likely that a large-screen display device would be an attached assembly to the primary processor.

3.2.4.15 Expert Systems

This technology is applicable to the majority of the tasks discussed in this subsection of the report. The most obvious application is to the task of supporting command decision-making. It has been successfully applied to pattern recognition and image scene analysis tasks. The field of expert database systems is currently an important research area; the knowledge gained from the many research projects could prove of great benefit to the BM/C3 community. It is difficult to make a general statement about the applicability of NvN architectures. The majority of extant expert systems were designed as standalone systems, but there is currently considerable activity in designing expert systems that are embedded into other information systems. The fruits of this activity will, no doubt, indicate how the various NvN architectures might be applied to the BM/C3I arena.

3.2.4.16 Message Processing

It is conceivable that with a Class VI machine (Multiple-PE architecture), separate processing elements could be assigned to the processing of different types of messages. In addition, expert systems technology could be an integral part of the message processing function.

3.2.5 The SDS Battle Management/Fire Control Functions for Space Based Processing

The previous subsections treated BM/C^{3}I functionality in a generic fashion. In this section a particular subset of BM/C^{3}I functionality is discussed; specifically, the system operations and integration functions of the SDS Battle Management/Fire Control Functions for Space-Based Processing.

The processor sizing estimates given here for the SDS Space-Based Battle Manager represent a first step toward characterizing the algorithms of an existing BM/C^{3}I system leading, in the next step, to an analysis of parallelizability and the mapping of parallel algorithms and algorithm segments onto particular NvN architectures.

These processing estimates assume decoupled boost post-boost engagement from mid-course engagement. The Space Based Interceptor (SBI) Constellations are being controlled by SAKTA Platform-based battle managers. The functions included here are those associated with system operation and integration only; signal processing and communications processing are not included.

Within the signal processing capabilities of each sensor, scan-to-scan correlations are made. Each sensor outputs angle data with correlations to previous scans. The functions discussed and sized here take such correlated data, perform track initiation in boost phase, use multi-sensor BSTS data to propagate tracks through boost phase, associate SSTS data in late boost phase and use SSTS sensors for PBV tracking. In addition, it is assumed that SSTS sensors will be used also to track interceptors flying out from SBI platforms.

This section discusses all the SIOP functions associated with boost and post-boost phase engagements. It is assumed that these functions are being performed by each appropriate SATKA platform implicitly, and that messages are sent to assigned SBIs by selected platforms.

It is difficult to estimate the processing functions because different functions independently make assumptions of worst-case loading. In addition, because the functions might be partitioned to different processor elements it is not clear that the worst case processing load actually occurs when the total processing reaches a maximum. The estimates given here are based on heavy loading for each of the functions. If actual parameter values were used here, this report would have to be classified. Assuming the review and discussion will be conducted in a non-classified mode, we make use of parameter values that are accurate enough, but which avoids a necessity for classification. The parameters are identified in Table 3-3.

Table 3-3. Parameters Used in BM/C3I System Sizing

I	Number of interceptors in flight
M	Missiles launched in a three-minute period
W	Number of weapon platforms in constellation
A	ASATs launched in the same three-minute period
S	Number of sensor platforms performing computations
Ta	Assignment time
Ts	Sensor scan time
Tf	Fire control update period

Assuming a very heavy mass raid in which the number of missiles launched in a three-minute period may vary from 500 to 2000, the number of interceptors launched from SBI platforms is given by the following formula:

{interceptors-launched .LTE. ([(# platforms_in_the_battle) * 20%]*15)}

The number of SBI weapon platforms in the constellation is of the order of a few hundred, say 300. Of these few hundred about 20% may be in the battle, and each of this 20% may be attacked by, say, three ASATs. The number of sensor platforms is likely to be fewer than thirty. Many weapon-to-target assignments take ten seconds. The scan time depends upon the sensor and will be less than ten seconds; a good nominal value is five seconds. The fire control update rate is between five and thirty seconds. An estimate of the number of computations to be performed is given in Table 3-4.

Table 3-4. Estimates of the Number of Computations per Function

Functions	Number of Computations (1000s of Operations)	Repeat Every X Seconds	Can be Parallelized
Sensor/Comm Assignments	O+2*S+0.5*(M+A)+0.1*I	30	yes
Statusing	5+0.1*(W+S)	60	
Weapon Update	1+2*W	30	
Sensor Processing			
Receive Data/Validate	0.15*(M+A+I)	Ts	yes
Correlate	0.3*(M+A+I)	Ts	
Track Initiate	0.6*(M+A)	Ts	
Track Propagate	2*(M+A)+0.8*I	Ts	
Type Discriminate	0.1*(M+A)	Ts	
Weapon Target Assignment			
Engagement Opportunities	3*A+1.2*(M+A)*2*W*(M+A)	Ts	not easy
Optimization	1.2*(M+A)*W*0.8*(M+A)	Ta	
Fire Control			
Engageability	1000*(0.05)*(M+A)	Ta	
In Flight Analysis/Homing View	1.2*I	Tf	

The first column of Table 3-4 names the functions; the second column contains algebraic formulae for deriving the number of computations; the third column gives the repetition rate for each function; the fourth and last column indicates whether a function's algorithms can be reexpressed in a parallel form to take advantage of NvN parallel architectures. To obtain an estimate of the number of computations per second, divide the results yielded by the formula in column two by the factor (either a numerical value or a parameter identifier) in column three. In general, those formulae that contain terms that sum the parameters "M" and "A" can be parallelized into separate passes for each missile and each ASAT. Algorithm optimization is a major exception to this general rule, however. Similarly, terms containing the parameter "I" are candidates for parallelization.

Estimates of the sizes of the databases used by these functions are given in Table 3-5. The first column identifies the databases; the second column gives a unit-size estimate; the third column gives the database update rate (either a numerical value or a parameter identifier); the fourth column identifies the functions that access the database; the final column indicates whether the functions read, write, or read and write the database. Key concerns in using an NvN architecture for a BM/C³I system are the matching functions and the data upon which they operate. For example, the sensor and weapon status tables, the track file, and the interceptor files all must be protected in nonvolatile memories.

Table 3-5. Database Size Estimates and Usage Identification

Database	Size	Update Every X Sec	Used By	Activity
Sensor Status	200 Bytes/Sensor	60	Sensor/Comm Asgts Statusing Receive Data/Validate	(R,W) (R,W) (R)
Weapon Status	400 Bytes/Weapon	30	Weapon Status Weapon Updates Engagement Opportunities	(R,W) (R,W) (R)
Track File	320 Bytes/ Missile & ASAT	Ts	Receive Data/Validate Correlate Track Inititate Track Propagate Type/Discriminate Engagement Opportunities All Fire Control	(R) (R) (R,W) (R,W) (R,W) (R) (R)
Interceptor	400 Bytes/Interceptor	Ts	All Fire Control Track Propagate Corelate	(R,W) (R,W) (R)
Sensor Data Buffer	3(40 Bytes) * (M + A + I)	Ta	Receive Data Track Inititiate Track Propagate Correlate	(R) (R) (R) (R)
WTA Working Store Tree	300 KBytes		Engagement Opportunities Optimization	(W) (R,W)

3.2.6 What BM/C^{3}I Systems Will Look Like in the 1990s

The CSC Team's investigations to date have not revealed any examples of NvN architectures applied to existing and operational BM/C^{3}I systems. However, the literature indicates a substantial amount of research and development experimentation with NvN machines for several of the typical data processing/computational tasks that characterize BM/C^{3}I systems.

The May/June 1988 issue of Defense Computing contains an article that discusses the nextgeneration of architectures for Electronic Warfare systems [Seals 1988]. The gist of this article is that EW systems of the future will incorporate a variety of architectures because no single architecture can efficiently handle all the technical problems. It seems reasonable to assert that we can extrapolate from EW systems to the larger context of BM/C^{3}I systems in general and say that BM/C^{3}I systems, at least in the 1990s time frame, will be comprised of networks of federated clusters of processors that are likely to be instances of virtually all the NvN architectures, operating in conjunction with traditional von Neumann machines.

Particularly within the context of distributed BM/C^{3}I systems, it seems unlikely that NvN machines will stand alone as hosts for complete BM/C^{3}I applications. It is much more likely that they will be key components of complex hardware aggregates made up of NvN, TvN, and possibly even analog machines. BM/C^{3}I system requirements, in general, are complex enough that separate, large-scale application programs, such as those for FORCE management, are likely be implemented on the hardware aggregates and not on single machines, whether NvN or TvN.

Assuming the continuation of current Air Force doctrine, it is likely that the BM/C^{3}I systems of the 1990s will be implemented on local area networks (LANs), with the nodes of the LANs likely to be the complex aggregates of machines mentioned above. Although NvN machines embedded within LAN nodes offer faster or more efficient computation of some BM/C^{3}I algorithms, the price paid for better speed or efficiency is increased complexity in manipulating data passed between the various types of machines within any particular computational aggregate.

3.3 ARTIFICIAL INTELLIGENCE

3.3.1 Introduction

3.3.1.1 Overview of Artificial Intelligence Production Systems

Artificial Intelligence (AI) production systems are rule-based programs that execute in a iterative manner. The principal components of a Production System (PS) are:

- a working memory (WM) that constitutes a data base for the system,

- a set of rules that correlate particular states of the working memory with actions to be

performed (including working memory changes),

- a driver program that iteratively evaluates the applicability of rules and performs actions associated with firing selected rules' actions.

Although Production Systems can be programmed in both general purpose and a variety of special purpose languages, the fundamental form of a rule, or production, is:

Label: $(c_1, c_2, ...c_n) \rightarrow (a_1, a_2, ...a_n)$, where "Label" is a unique identifier for the production, $(c_1, c_2, ...c_n)$ are boolean conditions, which reference WM contents and which must all be true for the rule to be applicable, and $(a_1, a_2, ...a_n)$ are actions to be performed when the all the associated conditions are true. Actions typically involve changes to WM contents and IO operations.

Production Systems can serve as the basis for different kinds of programmed systems (e.g., forward and backward chaining reasoning systems). The fundamental model for PS iterative execution is:

```
initialize working memory;
REPEAT
        FOR all rules
                evaluate conditionals;
                IF all conditionals are true,
                THEN add rule to conflict set;
        END_FOR;
        select one or more rules from the conflict set;
        perform WM updates and other actions specified by selected rule(s);
UNTIL (halting condition is encountered).
```

Note that expert systems are Production Systems that contain rules derived from human experts. Since this study's focus is the relationship between architectural features and PS performance it will focus on Production Systems in general.

3.3.1.2 Production System Architecture Research

This section describes recent research into multiprocessor architectures specifically geared to supporting AI production systems. Note that architectures designed to support the LISP programming language [Hwang, et al., 1987] are much more general in intent and are not reported here.

1. Production System Parallelism Research (Carnegie Mellon Univ.)

Carnegie Mellon University (CMU) research with significant implications for PS architecture issues has included: PS algorithm research, studies of fielded PS execution characteristics, and investiga-

tions of architectural features for PS application efficiency.

a. Rete Algorithm

The Rete algorithm, first proposed by Forgy [Forgy 1982] and later modified by Gupta [Gupta 1984], compiles PS specifications into a dataflow graph in which rules (productions) that share conditions share graph nodes that ascertain whether those conditions are met. Tokens, which consist of an add or delete tag and an ordered list of WM elements, are propagated through the graph during each PS cycle. Rete graph nodes consist of [Gupta 1987]:

- constant test nodes determine whether a WM element has a given constant value;

- memory nodes store tokens indicating the results of previous match attempts: 'alpha memory nodes' storing individual match test results and 'beta memory nodes' storing the match test results of conjunctive tests;

- two input nodes test for joint satisfaction of conditions and consistent variable bindings;

- terminal nodes indicate whether a production should be added or deleted from the conflict set.

Rete is a modest state saving algorithm, in that the results of previous cycles' match attempts for single condition tests and some conditional conjunctions are kept, but not the results of whether every permutation of a rule's conditions were matched or not. The Rete algorithm provides a degree of parallelism by letting rules that share conditions also share graph nodes where the conditions are evaluated. A major source of Rete efficiency springs from minimizing computations that accrue from WM changes by propagating only the tokens associated with affected productions through the graph during the next cycle.

b. PS Measurements

A detailed study of fielded Production Systems [Forg 1981] written in the OPS5 PS language and various simulation projects (see [Forgy, et. al. 1984] and [Gupta, et.al. 1986]) identified several important characteristics of such systems, including:

- a change to WM typically affects few productions (rules),

- coarse-grained production parallelism affords limited speed-up possibilities.

c. Architectural Research

As a result of these studies and simulations, CMU researchers have concluded that the fine-grained parallelism, moderate state-saving approach represented by the Rete algorithm is the most promising

direction for parallel PS development [Forgy, et.al. 1984], [Gupta, et.al. 1986], [Gupta 1987]. They have concluded [Gupta, et.al. 1986] that the most important architectural features for parallel Production Systems are :

- shared memory

- a modest number of high-performance processors (maximum of 32-64)

- the use of shared buses to connect processors to shared memory

- a hardware task scheduler.

While various ideas for a CMU Production System Machine have been proposed [Forgy and Gupta 1986] [Gupta 1987], current research is apparently focusing on using an Encore Multimax [Tambe, et.al. 1988].

2. Tree Topology Architecture Research (Columbia University)

The basic thrust of architecture research for PS applications at Columbia University has involved tree-structured multiprocessors utilizing low and intermediate capacity PEs.

a. DADO Architectures

The DADO [Stolfo 1987] and NON-VON [Shaw 1982] architectures developed at Columbia University were strongly shaped by the goal of efficiently supporting parallel PS operations [Stolfo and Miranker 1986] [Shaw 1985] [Shaw 1987]. The two architectures are also both characterized by a tree topology interconnection network for processor to processor communications.

The DADO2 architecture is a 1023-processor machine that uses a special I/O switch and VLSI circuit to connect processors. DADO2 is a partitionable MIMD/(M)SIMD machine, in which PEs can operate in SIMD mode by effectively broadcasting instructions to subtree descendents or receiving instructions from ancestor PEs.

Several algorithms have been proposed for implementing PS on DADO [Stolfo 1984] [Miranker 1984] [Gupta 1987]. Three of these algorithms are outlined below as described in [Stolfo 1984]:

- original DADO algorithm—the DADO tree is divided into
    (1) an upper tree portion devoted to synchronization and conflict set resolution
    (2) a PM-level that operates in MIMD mode on a subset of the production rules
    (3) a WM-level that consists of PE subtrees that act in SIMD fashion as an associative memory under the control of an ancestor PE at the PM-level.

- fine-grained Rete algorithm—a Rete dataflow graph is mapped onto the DADO tree-structured architecture, which operates in MIMD mode with a natural pipelining effect.

- TREAT algorithm—also involves partitioning the DADO tree into upper tree, PM and WM levels. Rule conditions are treated as relational algebra terms that are tested at the WM-level. The TREAT algorithm saves alpha memories (results of single condition tests) in the WM subtrees, and computes only those beta memories (results of two condition tests) that changes to the working memory (PS database) indicate will be relevant to the next cycle of computation.

Although Gupta has estimated the maximum performance of DADO PS algorithms at 215 working memory element changes per second (WMECS) for the TREAT algorithm and 175 WMECS for the Rete algorithm [Gupta 1987], the most recent Columbia data available for this study [Stolfo 1987] does not give any performance data in terms of WMECS.

b. NON-VON Architecture

The general NON-VON architecture [Shaw 1985] employs a complete binary tree of 8-bit small-processing-elements (SPEs) in which nodes are connected to tree neighbors (ancestors and descendents), as well as to other nodes at the same tree level. One or more microprocessors, or large-processing-elements (LPEs), are connected to various parts of the binary tree and can control the small processing element subtrees beneath them in SIMD fashion.

Shaw has estimated NON-VON PS performance using a parallel version of the Rete algorithm [Shaw 1985]. The simulated NON-VON algorithm used two parallel SIMD steps to perform intra-condition testing: first, simultaneously evaluating individual terms in rule conditions, then determining whether conditions' relational operators were satisfied and whether variables appearing more than once within a single condition were bound consistently. Inter-condition testing (determining whether a variable appearing in multiple conditions associated with a single rule is bound consistently) was performed by multiple-SIMD execution, in which LPEs used associated SPE subtrees to perform an associative search.

Using estimates for non-overlapped LPE execution and an instruction level simulator for SPE operations, Shaw projected that a NON-VON configuration with 16K SPEs and 32 LPEs could execute production systems at 903 rule firings per second or 2000 working memory changes per second [Hillyer and Shaw 1984, Shaw 1984, Gupta 1987].

3. Data-Flow Architecture Research (Honeywell)

Researchers at the Honeywell Computer Sciences Center and at the University of Kaiserslautern in West Germany have proposed a data-flow architecture, PESA-1, to support parallel PS execution [Ramnarayan, et.al. 1986]. This approach is predicated on mapping the data-flow network used by

the Carnegie-Mellon Rete algorithm onto a bus-based data-flow architecture. In such a scheme, PEs perform the functions of Rete nodes testing equality with constants, checking variable bindings, storing WM elements that have met prescribed conditions, and storing instantiations of PS rules to be added to or deleted from the conflict set.

The proposed PESA-1 architecture is structured as multiple levels of processors and memories (numbered 0 through n) with buses connecting adjacent levels. Each level, i, of processors and memories is connected to three buses (i-1, i, and (1+i)mod n+1). This connectivity rule ensures that level n 'wraps' back to level 0. PEs at level i can send their outputs to any of these three buses. Nodes that are at the same level in the Rete algorithm's data-flow network are mapped to these PESA-1 physical levels.

Rete tokens (see previous section on CMU research) propagate downward through PESA-1 levels, in a manner analogous to their propagation through the Rete graph. When a token reaches a given PESA level, it is broadcast to all the PEs at that level, which check a field within the token record that indicates whether the token should be processed at that level or forwarded to the next level. A token may be processed by all the PEs at a given level (i.e., used to check the consistency of variable bindings in a rule's conditions); however, only one PE stores that token for use in subsequent cycles. The authors suggest various schemes for determining the storing PE in a way that achieves uniform load distribution [Ramnarayan, et.al. 1986].

Synchronization is accomplished by having all PEs at a given level communicate that any required processing of the current token has been completed before work on the next token is begun.

4. Associative Memory Architecture Research (Loral Systems Group)

Research performed by Loral Systems (formerly Goodyear Aerospace Corporation) [Reed, Smit, and Lott 1986] suggests that the parallel processing capabilities of an associative memory architecture can be effectively utilized for expert systems.

Reed describes implementing a production system for real-time ELINT operations on the ASPRO, a militarized version of the STARAN associative memory architecture. This PS consisted of 545 production rules and 582 facts. Performance results were reported for a simulated tracking scenario. During the 10 real-time scenario minutes, 2 seconds of ASPRO compute time were consumed to perform : 4637 parallel searches of the rule data base, 6524 rule firings, and 2164 track report responses. The ASPRO, therefore, achieved 1.2 million rule interpretations per second. Note that if the 6524 rule firings involved an average of 3 WM (data base) updates per rule, the system achieved 9786 working memory element changes per second.

A key aspect of the ASPRO PS implementation consists of using bit-slices to represent rule conditions, rule consequences, and the current state of the working memory (WM) in order to exploit the parallel processing capabilities of an associative memory architecture. Preprocessing operations

construct horizontal bit-slice representations of rule preconditions and results. The current state of the working memory, is represented at run-time by vertical bit-slices. Each rule bit-slice is compared against the current WM bit-slice in parallel, generating a mask which flags rule preconditions that are not matched in the WM. A single OR instruction is used to update the WM bit-slice by setting the bit positions corresponding to the 'assertions' associated with rule firings.

Several implementation constraints should be noted. First, the system is a closed domain, in that new data base components cannot be added dynamically; this is essentially a consequence of mapping projected database values to bit-slice locations during preprocessing. Second, the number of PS rules is restricted by the number of ASPRO processing elements (1792, in the reported application). Finally, the data input to the system in real-time must be translated into "domain expressions" that can be mapped to the proper data base bit-slice position, although this operation can be performed by preprocessors associated with the ASPRO.

3.3.2 Production System Applications Characterization

3.3.2.1 Fundamental Processes

Artificial Intelligence Production System applications involve the following fundamental processes:

- initialization—The working memory (PS database) is initialized to some appropriate state; if a state-saving algorithm is employed, a data-flow graph or equivalent data structure must be initialized, possibly involving considerable computations.

- data acquisition—If the application accommodates WM changes from sources other than rule firings (e.g., the ASPRO tracking and surveillance application [Reed, et. al., 1986]) then the acquisition of WM data, possibly in real-time, will constitute a significant application process.

- condition evaluation—In order to determine which PS rules are applicable, the system determines whether the preconditions associated with a rule have been met. Such conditions are typically expressed as a list of boolean relations. Algorithms that save previous state information do not have to evaluate each condition on every pass through the execution cycle.

- rule selection—Rule selection involves determining whether all the conditions associated with a rule have been evaluated as true; thus, the selection process involves all the individual instances of condition evaluation.

- conflict resolution—When more rules may be selected as applicable than can be fired in a single cycle, one or more rules are selected from the conflict set to have their associated

III-30

actions executed. A variety of strategies can be used for resolution, including most recently fired, less recently fired, and user-defined priority.

- rule firing—The rule firing process involves performing the actions associated with a production. Typically, these actions are either WM value changes or I/O operations.

3.3.2.2 Key Algorithm Types

1. Algorithm State Saving Characteristics

PS algorithms may be characterized according to the degree of state saving (storing results of previous condition evaluations) that they exhibit [Gupta, *et.al.* 1986]. A spectrum of possibilities exists that runs from no state saving through TREAT, Rete and Oflazer algorithms. These algorithms are outlined below in ascending degree of state saving.

- A non-state saving algorithm does not store information about condition evaluation from previous cycles.

- The TREAT algorithm [Miranker and Shaw 1984] stores information about individual condition matches against WM elements.

- The Rete algorithm saves information about both individual condition matches and some combinations of condition matches.

- The algorithm proposed by Kemal Oflazer [Ofla 1987] would store information about most relevant combinations of condition tests for each production.

2. Algorithm Execution Modalities

Reported PS algorithms may be classified according to their execution modality, as shown below:

a. MIMD

The parallelized Rete algorithm developed by CMU researchers [Gupta 1987], [Gupta 1988] is an MIMD algorithm in which PEs concurrently perform the various kinds of processing associated with the Rete dataflow graph nodes. The fine-grained Rete algorithm for DADO architectures reported by Stolfo [Stolfo 1984] maps the Rete graph onto the DADO tree structure for MIMD operation.

b. MIMD/(M)SIMD

The original DADO algorithm reported in [Stolfo 1984] exhibits MIMD/(M) SIMD execution, with PEs in the upper tree performing synchronization and selection operations in MIMD mode, while PE

subtrees at the WM-level perform independent SIMD associative memory searches. The TREAT algorithm [Stolfp 1984], [Miranker and Shaw 1984] exhibits similar MIMD/SIMD execution.

c. (M)SIMD

Shaw's NON-VON OPS5 algorithm [Shaw 1985] uses 32 large-scale processors to control (M)SIMD processing of Rete inter-condition testing, but does not appear to utilize MIMD processing.

d. SIMD

The PS algorithm for the ASPRO associative memory processor [Reed, Smit and Lott 1986] uses SIMD execution to simultaneously compare bit-vectors, which represent rule conditions, against a bit-vector representing the current Working Memory state.

3.3.2.3 Performance Requirements

PS performance requirements are clearly shaped by the particulars of a given application such as number of rules, working memory volatility (average database updates per rule firing), and whether real-time response is required. In addition, the environment (e.g., land-based, embedded, space-borne) can reasonably be expected to influence performance requirements. Although precise evaluations of PS algorithm performance requirements would have to be made on a case-by-case basis, meaningful judgements about architecture suitability for PS performance can be drawn by looking at the essential components of PS algorithms that will influence performance.

Published studies of PS performance requirements (e.g., [Gupta, et.al. 1986], [Quinlan 1986]) have identified loads, compares and branches as the instructions most often involved in PS execution. Note, however, that even meticulous studies are sometimes shaped by presuming the superiority of a particular algorithm or by assuming that an architecture must possess TvN machine features, such as general-purpose registers. For example, the performance of a PS algorithm running on a representative TvN uniprocessor would be heavily influenced by the comparative speed of operations such as loading a pair of registers with a WM element and an element appearing in a rule condition, and then comparing them. However, as the Loral ASPRO PS approach demonstrates, it is possible to implement a PS in which traditional register load and compare operations play no significant part. Performance analysis for this study, therefore, will be couched in terms of generic processes, rather than instruction types that might involve unduly restrictive presumptions about architectures.

The most important general performance requirements for a PS are:

- data acquisition—For some applications, such as a PS requiring real-time response and handling a heavy computational load due to a large number of conditional evaluations or

WM updates, it is likely that PEs must be available to prepare incoming data for WM insertion. Note that potentially complex synchronization may be required if the same WM components can be changed by both rule firing and incoming data.

- condition evaluation—Boolean tests must be conducted rapidly, with as much parallel evaluation as is productive. It is likely that some state saving will be employed to reduce the number of evaluations required per cycle. Note that this process is likely to be the most critical to overall system performance [Gupta, et.al. 1986] and the process that most requires fast memory subsystem performance.

- rule selection—Atomic conditional evaluations should lead to identifying relevant rules as soon as possible; it is desirable to avoid having the execution time for determining each rule's relevancy be proportional to the largest number of preconditions associated with a rule.

- working memory updates—Although the relevancy of few productions may be altered by the updates triggered by a single rule's firing ([Gupta, et.al. 1986], [Gupta 1987]), the performance of this process may be critical in the face of either distributed copies of WM (hence, updating multiple WM copies) or of synchronization measures that allow WM contents to be updated by both external data acquisition and rule firings.

3.3.2.4 Hardware Architecture Demands

Just as the PS performance analysis above has been conducted in generic terms to avoid precluding consideration of innovative architectural or algorithmic solutions, hardware architecture demands will be analyzed in a similar manner. Likely PS hardware architecture demands are reported below.

a. Memory Demands

Memory subsystem speed is likely to be critical to overall PS efficiency because matching production preconditions to the current WM contents consumes the vast majority of compute time in studied applications [Gupt, et.al. 1986]. This aspect of PS performance has encouraged approaches using both associative memory processors [Reed, Smit, and Lott 1986] and subtrees of low-capacity processors with private memory to essentially mimic associative memories (e.g., [Shaw 1985], [Stolfo 1984]). Shared memory solutions (such as [Tambe, et.al. 1988]) will place heavy demands on access synchronization and cache coherency mechanisms.

b. Processing Demands

Because condition testing consumes more of the execution time in surveyed PS systems, the speed of comparison instructions is critical to PS applications. In more traditional architectures, this factor militates for scalar processors with fast boolean test instructions; the Loral associative memory approach suggests bit-vector comparisons as a possible alternative.

c. Interconnection Network Demands

Reported results suggest that processor-to-memory interconnections are not a performance bottleneck for PS applications using a Rete-style algorithm, a small number of processors, and shared memory [Gupta 1987], [Tambe, et.al. 1988]. However, in cases where the WM is partitioned and distributed among multiple memories, processor-to-memory interconnection performance could become a limiting factor.

The Columbia University research described above involved tree structured processor-to-processor interconnections. If a dataflow PS approach is used in conjunction with such an interconnection topology, the local nature of data flows is likely to prevent the interconnection network from constituting a performance bottleneck. In the event that the processor-to-processor interconnection topology does not so closely match the algorithmic approach, interconnection network latency could become critical to required performance.

d. Environmental Demands

In addition to the performance demands that accrue from the computational characteristics of PS applications, special demands on hardware architecture may be dictated by environmental requirements. For example, planned spaceborne applications will have to meet demands for ruggedized, radiation-hardened components [Lum 1988].

3.3.3 NvN Architecture Suitability for AI Production Systems

The following assessments of architecture suitability for AI production systems are based on both reported research results and conceptual analyses. These assessments do not rely on reported timing data in an absolute sense, because such timings are often based on software simulations whose precision cannot be established and because timing data is reported in diverse forms that cannot be compared fairly.

3.3.3.1 Pipelined Vector Uniprocessor Architectures (Class I)

The suitability of Pipelined Vector Uniprocessor Architectures for implementing production systems appears to be contingent on the development of "vectorized" algorithms for production condition testing. While the fast scalar processors found in most such architectures may achieve better PS performance than some multiprocessor architectures, this could not, in itself, make vector architectures optimal for PS. Preliminary results achieved by Loral with a bit-vector, associative memory processing approach to production systems, however, suggest that effective PS algorithms based on boolean or integer vectors might be a fruitful area for research.

3.3.3.2 Rhythmic Cellular Control Architectures (Class II)

Since Rhythmic Cellular Control architectures (systolic and wavefront arrays) are geared to implementing predictable, orthogonal calculations, they are not well-suited to the condition testing

phase that is most important for efficient PS implementation. A Rete-style dataflow algorithm could conceivably be mapped onto a two-dimensional systolic or wavefront array, assuming that appropriate operands (e.g., tagged tokens and a list or mask of WM elements) could be pulsed from PE to PE. However, other architectures are likely to provide more cost-effective solutions for PS implementation.

3.3.3.3 Processor Arrays (Class III)

Processor Arrays are not promising candidate architectures for implementing most AI production system algorithms. The classic SIMD mode of processor array operation is not well suited to the condition testing operations that constitute most of a production system's computational demands. Two significant problems arise from processor arrays broadcasting a single instruction to multiple PEs that operate in lockstep. The conditions to be tested may involve different boolean tests, such that only a few PEs can operate in parallel for each cycle. Second, since productions are associated with a varying number of conditions, the time expended on condition testing would be proportional to the length of the longest list of conditions. Condition test sequences for productions could be altered at compile time to make all test operations using the same instruction appear in the same order for each production. However, the time taken for each type of test would be proportional to the longest list of tests of that type associated with a production.

Although Forgy explored implementing a PS on the Illiac-IV [Forgy 1980], it does not appear that any hard timing data or especially promising techniques resulted from that effort [Gupta 1987]. A radically different algorithmic approach might make Processor Arrays more suitable for production systems, but current PS techniques are not amenable to efficient parallel implementation on Processor Array architectures.

3.3.3.4 Associative Processor Architectures (Class IV)

Associative processors appear to be a viable architecture for the kind of bit-vector oriented algorithm developed by Loral [Reed, Smit and Lott 1986]. Such architectures achieve considerable parallelism in the condition testing phase, since a large number of processors can operate in parallel to test bit-slices representing conditions against a bit-slice representing the current working memory's contents. Note that rule firing that alters WM contents can essentially be executed with a single OR instruction with this approach.

Although preliminary results are encouraging, at least two significant constraints should be noted. First, the number of PEs effectively constrains the number of productions that can be accommodated. Since current research involves more than 2000 PEs this limit does not appear to be an onerous one, however. Second, the bit-vector approach requires that WM elements be reduced to a single piece of information. In the embedded, real-time environment for which the Loral ASPRO PS was designed, this requires that the data acquisition process perform a substantial amount of data compression. A heavily loaded associative processing PS, therefore, may require a set of preproces-

III-35

sors to cast real data into a bit-vector format. Despite these limitations, reported research suggests that associative memory architectures are legitimate candidates for parallel PS implementations.

3.3.3.5 Operand-Driven Architectures (Class V)

The dataflow (data-driven) subclass of operand-driven machines is a suitable architecture for implementing PS algorithms based on a dataflow model of execution, such as the parallelized Rete algorithm [Gupta 1987]. Although the viability of the particular architecture proposed by Honeywell researchers [Ramnarayan 1986] is likely to depend on both bus latency and load balancing techniques, the structural match between data-flow architectures based on packet communications and data-flow PS algorithms makes this architectural subclass an attractive candidate for PS implementations. Note that the reduction (demand-driven) subclass of operand-driven architectures is not as promising, since its computational model based on nested expressions does not closely match PS condition testing operations.

3.3.3.6 General-Purpose Multiple-PE Architectures (Class VI)

Given the diversity of General-Purpose Multiple-PE (GPMPE) architectures, this section will separately evaluate the GPMPE subclasses for which timely research results are available.

a. Bus-Based Processor-to-Memory Interconnection Architectures

The largest body of published PS architecture studies reflects the work of Carnegie-Mellon researchers (e.g., [Forgy, et.al. 1984] [Gupta, et.al. 1986] [Gupta 1987] [Gupta, et.al. 1988]). Their metrics for existing PS applications written in OPS5 suggested that:

(1) a PS offers limited opportunities for parallel execution speed-up;

(2) exploiting PS parallelism at a fine-grain level is most effective; and

(3) working memory changes caused by rule firing affects relatively few productions.

As a result of these findings, they have argued that a shared memory, bus-based architecture with roughly 16-64 processors is an extremely effective architecture for production systems implemented with a Rete-style algorithm. In support of this conclusion, they argue that relatively few processors (generally 10-12) can be kept busy with productive work, that shared memory and bus contention does not appear to be a bottleneck, and that contention for the scheduler PE has the most practical impact. In sum, they make a compelling case for the suitability of this architecture type, at least for known PS applications written in an OPS5-like language and implemented with a Rete-style, data-flow algorithmic approach.

b. Tree Structured, Processor-to-Processor Interconnection Architectures

To date, evaluations of this architectural type have essentially been made on the basis of predicted performance for the DADO and NON-VON architecture families developed at Columbia University [Gupta, et.al. 1986, Hillyer and Shaw 1984]. These predicted performance levels (approximately 900 rule firings/sec. and 2000 WM changes/sec.) fall short of predicted results for both bus-based shared memory and associative memory architectures. Several mitigating factors should be taken into account, however. First, the modest power of existing tree architecture PEs (.5-3 MIPS) may be an inaccurate gauge of that architectural type's potential performance. Second, most of the reported data assumes that the PS is coded in OPS5 or OPS83, that a Rete-style algorithm is employed, and that the PS possesses structural relationships similar to fielded OPS5 PSs. Hence, Gupta [Gupta 1987] suggests that the tree architectures might show more promising performance for production systems characterized by a much larger 'affect set' (number of rules affected by a single rule firing's working memory changes). Stolfo [Stolfo 1987] reported significant speed-ups for PS implementations on DADO2 compared to uniprocessor implementations; however, metrics showing performance superior to other NvN implementations has not been available for this study. In sum, this type of architecture appears promising for PS implementation, but results have not yet been shown for such an architecture based on powerful PEs and using an algorithmic approach that fully exploits the architecture's characteristics.

c. Hypercube-Structured, Processor-to-Processor Interconnection Architectures

This section considers two kinds of research conducted for GPMPE architectures based on processor-to-processor interconnections with a hypercube topology.

A recent study [Gupta and Tambe 1988] concludes that message-passing, hypercube-structured architectures with decreased communications delays are effective hosts for Rete-style PS algorithms. The studied scheme partitions PEs into the following groups: control processor, PEs that perform constant-tests, PEs that perform conflict-resolution, and PEs that perform matching (variable binding) tests. Since checking variable bindings at "two-input" nodes of the Rete dataflow graph is the most compute-intensive part of the algorithm, a significant aspect of the proposed hypercube scheme involves using a distributed hash table to access items to be matched and using more than one PE to perform tests associated with a single Rete node. The study makes a cogent case for the suitability of message passing architectures. It should be noted that much of the case is based on recent increases in communication bandwidth and that 4 MIP processors are assumed.

Research performed by Thinking Machines Corporation on the Connection Machine, which exhibits a hypercube topology but is not a message-passing architecture, suggests that memory-based reasoning is an effective approach to problems in the PS domain [Waltz 1987, Stanfill and Waltz 1988]. The memory-based reasoning approach involves drawing inferences from a large database of individual cases, rather than using rule-based reasoning. Since timing data for this approach cannot be reported in terms of production firings/second or WM element changes/second it is

difficult to directly compare the performance of memory-based reasoning and PS applications. Other recent research [Blelloch 1986] suggests that the Connection Machine can effectively be used to implement an inferencing network model that provides functionality similar to PS capabilities. It is worth noting, then, that hypercube architectures are promising hosts for algorithmic approaches that attack problems similarly to the way a PS does.

3.3.3.7 Neural Network Architectures (Class VII)

Neural network architectures are best considered as possible alternatives to AI production systems, rather than as possible hosts for them. Consequently, no attempt has been made to evaluate the suitability of neural network architectures for implementing production systems.

3.3.4 Ranking NvN Architecture Classes on Their Suitability for Artificial Intelligence Production Systems

This section of the report ranks the NvN architecture classes that have been reported in recent technical journals as being viable for efficiently executing artificial intelligence PS algorithms.

Preceding paragraphs of this section of the Final Report discussed the analysis of NvN class suitability for the AI PS application domain. This section summarizes that discussion and presents a ranking of the most promising NvN architecture classes for AI PS. Section 3.3.4.1 briefly reviews AI Production Systems; Section 3.3.4.2 identifies the NvN architecture classes found to be suitable for executing AI production systems; Section 3.3.4.3 presents a tabularized ranking of those architectures that were found to be suitable.

3.3.4.1 AI Production System Review

Production rules can serve as the basis for a variety of AI systems. For example, backward-chaining systems, which are often used for hypotheses verification, start with a WM state and iterate through a cycle that determines which rule preconditions would have permitted rule action firings that resulted in some given WM state. Forward-chaining systems initialize the WM, then iteratively fire one or more rules ad infinitum, or until a specified halting condition is encountered.

The evaluation of NVN architecture class suitability that follows is based on forward-chaining Production Systems, since the technical literature on parallel architecture PS applications is dominated by this kind of system and provides a sound basis for the analysis.

3.3.4.2 Identifying Suitable NvN Architecture Classes

The PS performance studies that have been reported for NvN architectures can be used to identify the architecture classes that are most promising for AI PS applications. Although one cannot use the performance characteristics of the particular architectures studied to accurately predict the PS

performance of each member of an entire NvN architectural class, one can use the published research to establish an informed estimate of the relative suitability of each class for AI PS.

Table 3-6 shows the NvN architectures that have been used in recent PS performance research and the NvNACS classes to which they belong. Table 3-7 shows the results of different research activities published in the literature. These published results indicate different kinds of performance measurements that were obtained by various mechanisms. Table 3-8 indicates the relative rankings of various NvN architectures. These are discussed below.

1. Ranking Best Fits

a. Limitations of Existing Performance Metrics

There are two significant barriers to using published performance data to perform head-to-head comparisons of NvN architectures to determine suitability for forward-chaining production systems. First, there is no widely accepted standard for PS performance measure. Architectural efficiency has been reported using different metrics, such as the number of rules fired per second, the number of working memory element changes per second (wmecs), the speed-up over various single processor implementations, as well as the percentage of theoretically possible speed-up achieved. Such divergency in performance measures precludes any straightforward architecture comparisons. Second, published performance data has been obtained through dissimilar means, including benchmark timings, software simulations, and theoretical estimates.

Table 3-6. NvNACS Classes in Recent PS Performance Research

NvNACS Classification	Architecture	Research
Class IV: Associative Memory Subclass: Bit-Serial	ASPRO	[Reed 86, Lott 87]
Class V: Operand-Driven Subclass: PE-to-Memory Comm. Order: Packet Communication	PESA-1	[Ramnarayan 86]
Class VI: G-P-Multiple-PE Subclass: PE-to-Memory Comm. Order: Bus Interconnection	CMU PSM Encore Multimax	[Gupta 86, 87] [Gupta 88]
Class VI: G-P-Multiple-PE Subclass: PE-to-PE Comm. Order: Tree Topology	DADO, DADO2	[Stolfo 84, 87]
Class VI: G-P-Multiple-PE Subclass: PE-to-PE Comm. Order: Tree Topology	NON-VON	[Shaw 82, 85]
Class VI: G-P-Multiple-PE Subclass: PE-to-PE Comm. Order: Hypercube Topology	·hypercubes	[Gupta, Tambe 88]

Table 3-7. Performance Metrics for NvN Architectures

Architecture	Algorithm	WMECS	Rules Fired per Second	Timing Mechanism	Source
ASPRO	Reed Bit-vector	x	3262	timed	[Reed 86]
ASPRO	Lott Bit-vector	5340	800	timed	[Lott 87,88]
CMU PSM	Parallel-RETE	9400	x	simul'd	[Gupta 86]
DADO	Parallel-RETE	175	x	est'd	[Gupta 86]
DADO	TREAT	215	x	est'd	[Gupta 86]
NON-VON	RETE	2000	x	est'd	[Gupta 86]
NON-VON	Parallel-RETE	x	903	sim & est'd	[Shaw 85]

3.3.4.3 NvN Architecture Classification Rankings

Although the diverse performance metrics and measurement techniques described above cannot serve as the basis for rigorous comparisons of individual architectures, they can be used to categorize how compelling an argument has been made for the suitability of various NvN architecture types. Table 3-8 shows a ranking of NvN architectures; that is, the best fits for AI production systems.

Table 3-8. Ranking NvNACS Categories for Parallel PS Suitability

Rank	NvNACS Class	Summary of the Rationale and Remarks
1	Class VI: GPMPE Subclass: PE-to-Mem. Comm. Order: Bus Interconnector	copious research literature; extensive simulations; best simulated 'wmecs' performance (see above); highly flexible
1	Class IV: Associative Memory Subclass: Bit-serial	best benchmarked 'wmecs' performance; algorithms have more limited flexibility
2	Class VI: GPMPE Subclass: PE-to-PE Comm. Order: Tree Topology	significant speedups over uniprocessor implementations have been demonstrated
3	Class V: Operand Driven Subclass: Data driven Order: Packet Comm.	promising paper studies; evaluation must await implementation
3	Class VI: GpmPE Subclass: PE-to-PE Comm. Order: Hypercube Topology	promising paper studies; evaluation must await implementation

1. NvN Architecture Classification Ranking Rationale

Cogent arguments have been made for the suitability of both shared memory, bus-based multiprocessors [Gupta, et al. 1986, Gupta, et al. 1988] and associative memory architectures [Reed, Smit and Lott 1986, Lott 1987]. PS implementations exist for both kinds of architectures and benchmark data is available. In addition, both architectural solutions have shown significant speed-ups over uniprocessor implementations [Gupta, et al. 1988, Lott 1987] and both appear likely to achieve on the order of 9000 wmecs [Reed, Smit and Lott 1986, Gupta, et al. 1986].

A plausible case has been made for tree-structured multiprocessors using processor-to-processor communications [Stolfo 1987]. Production systems implemented on this architecture have demonstrated some rather significant speed-ups over uniprocessor implementations, but performance data comparable to that reported in the category above (in terms of wmecs or rules fired/second) has not thus far been published. Recent research suggests that significant gains in parallelism might be achieved by concurrently executing multiple production systems or WMs.

Performance estimates for both dataflow [Ramnayaran, et al. 1986] and hypercube message-passing architectures [Gupta and Tambe 1988] suggest that these architectural types are promising candidates for further research. However, until PSs have been implemented on these architectures and actual measurements have been reported, the case for the suitability of either architecture must be regarded as very preliminary.

2. Unranked NvNACS Classes

Several NvNACS classes or subclasses have not been ranked in the foregoing analysis. Lack of inclusion does not imply that these NVN architectural types would be inefficient hosts for forward-chaining production systems; it means, simply, that recent research literature does not include reports on this type of research using these architectures. The interested reader may refer to earlier paragraphs of this section for general remarks about the possible utility of these architecture classes for forward-chaining AI production systems.

3. Conclusion

Several NvN architecture types have recently been investigated as hosts for parallelized, forward-chaining PS algorithms. Available performance data for parallel production systems does not provide a definitive assessment of the suitability of all NvNACS architecture types for parallel PS applications. However, these research findings can reasonably be used to rank how compelling a case has been presented for the investigated NvN architecture types' viability.

3.3.5 Conclusions

This section summarizes the current state of research in applying NvN architectures to artificial intelligence production systems.

Research in PS architectures has been strongly shaped by the PS measurements performed at Carnegie Mellon University. This detailed investigation of PS metrics serves as a useful basis for comparing proposed architectures and algorithms. However, it is possible that future PS applications, especially those specified in a format that diverges from OPS5, may exhibit significantly different run-time characteristics.

Architecture comparisons are further complicated by two factors. First, in the absence of a standard performance measure, architectural efficiency has been reported in various forms that are difficult to compare, including number of rules fired per second, number of Working Memory (database) updates per second, speed-up over single processor performance, percentage of theoretically possible parallelism exploited, and percentage of theoretically possible speed-up achieved. Second, the performance data that is available ranges from actual timings, through simulations, to theoretical estimates.

An overall evaluation of NvN suitability for PS applications, therefore, must take into account different performance metrics, derived from divergent methods. Nevertheless, a general assessment can be made from the preponderance of available evidence. A strong case has been made for the viability of architectures characterized by a moderate number of processors (e.g., 16-32) that use bus interconnection technology to access shared memory. Preliminary results for associative memory architectures are impressive, within likely preprocessing constraints. Proposed PS architectures based on dataflow principles and on hypercube-structured message-passing appear promising, but there are no extant hard timing data that supports their viability. Tree-structured multiprocessors have shown some encouraging speed-up over uniprocessor performance, but appear to require either PEs with substantially more bandwidth or a different algorithmic approach to be fully exploited.

These assessments are clearly based on the current state of research, which is strongly influenced by both the prevalent dataflow algorithmic approach and the nature of fielded PS applications.

It is extremely difficult to foresee the emergence of radically different algorithmic approaches. Note also that new algorithmic approaches to implementing production systems may be accompanied by the development of techniques that exhibit PS functionality, such as memory-based reasoning or neural network learning, but that require very different architectural support. The current technical literature detailing the results of recent research certainly suggests that several of the NVN architecture classes can be exploited for more or less efficient parallel PS execution; however, forthcoming research might well extend the set of viable architectural solutions.

3.4 REAL-TIME SIMULATION

3.4.1 Introduction

This section of the Report describes systems designed to simulate North American air defense architectures and components of the Strategic Defense Initiative (SDI). The models, interactions,

and requirements detailed herein are intended to provide sufficient information on the structure and operation of the simulation to support the making of informed judgments regarding potential re-hosting of these simulations on Non-von Neumann (NvN) architectures. The simulation instances and the host architecture must be flexible and robust enough to accommodate models operating over a broad fidelity range, models not yet defined, as well as increases in overall load due to the need to simulate attacks with greater numbers of threats and weapons. The key to efficient, flexible, and robust simulation is a well-designed simulation executive. The executive is responsible for mediating the interactions of each model based on information from the user via the simulation control routine. Additionally, the executive must control the overall simulation to assure the repeatability of simulation events for the purpose of validating results. Design concepts for a simulation executive are discussed in the following paragraphs.

3.4.1.1 Simulation Executives

There are three generally used approaches to designing an executive for digital simulations of large systems: time-stepped, event-stepped (often called discrete event), or a hybrid that combines both these approaches. The executive is the key to successful simulation and therefore should be carefully designed, with particular attention given to simulation efficiency and repeatability in parallel processing environments. The following paragraphs briefly describe each type and identify facets or aspects of performance that should be considered during the design.

1. Time-Stepped Executive

With the time-stepped approach, the simulation is updated in discrete intervals, and time advances at a fixed rate based upon the interval selected. Interactions with objects in this type of simulation occur within a fixed number of intervals. The time-stepped approach provides an efficient synchronization mechanism and allows much flexibility. With the controlled interval, models may be distributed across multiple processors. Also, any level of model fidelity may be supported down to the resolution of the interval.

Discrete event models (e.g., communications models) can be incorporated into the simulation but are restricted to execute within the interval or a predefined finite number of intervals. Since the interval controls the run rate, periods of little interaction run at the same rate as periods of high activity. Simulation hardware and/or models must be sized to handle the worst-case load within the required number of intervals. This sizing constraint leads to system resources that are often idle under less than maximum load conditions.

2. Event-Stepped Executive

With the event-stepped approach, the objects in the simulation interact only at discrete points, and time leaps from one event to the next in time-sorted fashion. The event-stepped simulation, under some conditions, may be more efficient than the time-stepped simulation. Since time advances from

one event to the next in jumps, the simulation advances rapidly in periods of little interaction. Since the simulation is not tied to a fixed interval, each model may use any amount of time to process its load in simulating the system without impacting other models. This, however, does impact the total amount of time required to execute a run. Any level of model fidelity can be supported.

A master event queue centrally manages the simulation; therefore, exact event sequence repeatability is possible within a uniprocessor environment. The event-stepped approach forces limitations on the size of the simulation, and there exists no straightforward parallel/multiple processor implementation capability. The rate of run is dependent on the number of objects/models, and load and model complexity. The greater the number of objects and the number of models that interact with them, the greater the size of the processing queue and, hence, the more the efficiency of the executive is reduced and the longer the simulation takes to execute. This growth is a non-linear "m" or "n" function. Continuing study of industry technical literature indicates that there is not a general-purpose event-stepped architecture currently available for a multiprocessor environment.

3. Hybrid Executive

The Technical Evaluation Facility (TEF) for the Air Defense Initiative has many complex requirements that can be satisfied only by combining event-stepped simulation with time-stepped simulation. A hybrid executive could provide a centrally controlled discrete event simulation with an underlying selectable time period. The simulation would be structured, basically, as an instance of a discrete event design; however, event-time resolution would be limited to the time-step interval. The discrete-event executive advances time from event-to-event at an event-stepped or preselected rate. The event queue potentially would have many events for the same time interval that would be dispatched in FIFO order for a single processor configuration, or in parallel for a multiprocessor configuration. This would also allow the models to be made up of a mix of discrete and time-stepped processes and the objects could be updated in parallel with other activities. The primary disadvantage is that there is no hybrid executive available off-the-shelf. The runtime database must be very carefully designed to assure that data race conditions do not occur.

3.4.1.2 Simulation Repeatability in Multiprocessor Architectures

Repeatability of results is necessary to establish a level of confidence and to validate the results of the events occurring in the systems represented by the simulation. Simulation repeatability is defined here to mean that the results of any two simulations will have bit-for-bit fidelity, regardless of the time required to run a particular model.

Multiprocessor simulation environments generally have models running on several separate processors simultaneously, creating the potential for messages between processors (from simulated system elements) to be received at different times from run to run, thereby producing variable results. The following subsection discusses five methods to achieve the repeatability necessary to validate simulation results in a multiprocessor environment. These methods are: Scoreboard, Chandry-Misra, Timewarp, Management by Exception, and Precursor Messages.

III-44

1. The Scoreboard Method

With this method all messages are sent to a master list that determines when it is safe to run processes out of order. The method is based on the principle that physical laws or properties govern the behavior of objects under a given set of conditions. For example, consider the interaction of two processes S and R. Events resulting from process S require the passage of a certain finite amount of time before there can be an effect on process R, regardless of the actions of S. To be more specific, assume that S is the detonation of a nuclear weapon; its effects cannot propagate faster than the speed of light to affect process R. Events occurring in process R prior to the detonation and the necessary propagation delay are said to be outside the event horizon of process S. Within a battle management system, effects propagate at a rate limited by the supporting communication system. The executive in Scoreboard will run two processes out of order if and only if the later scheduled process is not affected by any earlier scheduled process.

Using this approach, the simulation begins by computing the event horizon of each process with respect to every other process. This is done by determining which processes have direct effects on any other process. At run-time, the executive maintains a list of all messages which are to be run. The decision to execute a message is made by identifying the processes currently executing and determining whether or not the new message is within the event horizon of the executing processes; if the new message is outside the event horizon, then it will be applied, assuming that a processor is available.

2. The Chandry-Misra Method

With this method, the time at which to send messages is determined from a message list that is maintained by each processor. Each processor also maintains a queue of messages for all other processors from which messages can be received. Assuming that each queue has at least one message, the processor picks up the message with the lowest timestamp. In the event that one of the queues is empty, the processor must wait in order to determine that an incoming message to the queue does not have the lowest timestamp.

3. The Timewarp Method

This method permits processors to execute messages up to a time-check-interval point. Each processor keeps only a list of the messages it receives and picks up the message with the lowest timestamp. Periodically, each processor makes a time check; if it finds a message with a timestamp that precedes the time check it must roll-back to the previous time check and issue a cancel-message order for every message up to the time the new message falls into sequence. At this point, normal processing is resumed. Note however that a roll-back of one processor may cause the roll-back of many other processors.

4. The Management by Exception Method

This method relies on each process to predict whether its output messages can be output in the time allowed. If the processor predicts it cannot meet the time deadline, it notifies the executive, which then halts all other processes until the slow processor can output its message.

5. The Precursor Messages Method

This method makes use of a "pre-message" message to notify a receiver that a message will be sent to it as well as the time when it should arrive. The receiving processor continues processing until the time it expects to receive the message and then halts processing until the expected message arrives; it then resumes its normal processing.

3.4.2. The Air Defense Model Environment

The ADI TEF models the North American Air Defense Environment and provides for interaction between simulated real-world objects and the simulated effects. This interaction takes place to simulate the outcome of events as they would occur under actual conditions. The model is defined in terms of the described attributes and parameters that determine the behavior of the simulated objects. The basic framework that links all the objects in the simulation is described in the following paragraphs.

The region of interest for the TEF is postulated to be a map area. Locations on the map and map boundaries are defined by latitude, longitude and altitude. The geographical area covered by the TEF simulation is the whole globe. The latitude-longitude-altitude (LLA) grid establishes the framework for representing object locations in a scenario as well as for defining environmental effects over the region of interest. Platform locations are defined by a point determined by the LLA values. Environmental effects, (atmospheric, electronic, or nuclear) are defined in terms of grid cells that are affected. The movement of platforms and environmental effects are governed by the models and parameters which define their behavior.

Sensors and weapons which may be affected by environmental conditions are modeled so that these conditions are accounted for within the grid framework. The path through which a sensor must look or a path on which a weapon must travel in order to detect or kill a target contains indexes within the grid which degrade the object's performance, e.g., the probability of detection (Pd), or probability of kill (Pk) under the simulated conditions.

The background grid of framework effects are recalculated periodically to account for the changes that may occur dynamically during a scenario, such as jammer movement or nudets. The use of latitude, longitude, and altitude as a grid system permits all objects and effects in the simulation to be described by a common reference frame, eliminating the need for stereographic conversions also reducing the number of machine instructions required.

3.4.2.1 Model Descriptions

A description is given, for each model postulated for the TEF, which includes recommendations for model interaction. The discussion is given in the following paragraphs.

1. Environment

The environmental model provides the background effects for the simulation. This model effects sensor detection capability, communication disruption, and also contains the effect of nuclear kills and interruptions. The environmental model divides the region of interest (15 degrees North latitude to the Pole and 360 degrees of longitude) into one degree latitude by one degree longitude grid cells. Each of these grid cells contains environment and nuclear derived effects which are updated by the environmental model.

2. Atmosphere

Atmospheric conditions considered in the model are cloud types and altitude, precipitation, humidity, and ionospheric activity. The conditions are entered as part of the simulation in grid cells one degree latitude by one degree longitude. The effects are considered constant over the region and may be scripted to change during the simulation.

3. Enhanced Environment

An enhanced environment is a result of nudets, and shows an increase in the background noise level detected by sensors and communication receivers. The enhanced environment effects are modeled on a grid of one degree latitude by one degree longitude, and show decreasing level and also the affected area slowly increasing over time. These changes require periodic recalculation on the order of once every five minutes or less.

4. Electronic

The effects of threat platforms equipped with ECM/ECCM equipment are modeled based on current cell location, jamming strength, jamming direction (omni or sector), and band.

5. Terrain

Terrain is stored as static map data and is used to determine sensor coverage holes and line-of-sight communication disruptions. The terrain mapping grid cell covers four minutes by four minutes.

6. Nuclear Effects

Nuclear effects modeled in the simulation result from salvage fusing, ADI leakage and SDI leakage. The effects modeled are nuclear bursts, fireballs, and the resulting enhanced environment.

7. Nuclear Burst

Nuclear bursts result in the destruction of defender assets, threats, and threat platforms within the lethal range of the nudet. The lethal range is computed based on the warhead yield and the Height of Burst (HOB). The time and location, in conjunction with the range, determine the grid cells affected by the nudet.

8. Fireball

Fireballs result in the disruption of sensor and communication system performance in the vicinity of the nuclear burst; the grid cells disrupted are determined by the yield and HOB.

9. Object Motion

The object motion models describe the movement and the operational characteristics of Threat, C2, Weapon, Neutral, and Sensor Platforms; each model type is discussed in the following paragraphs.

a. Threat Platforms—Threat platforms include submarines, surface ships, aircraft, and any other object which may pose a threat. This category also includes ECM/ECCM (Jammer) platforms which reduce or obscure sensor performance. Platform type characteristics are modeled with tables; these characteristics are: (as applicable)

> Platform type (submarine, surface ship, aircraft)
> Sub-designation (bomber, interceptor, etc.)
> *Flight profile/movement characteristics*
> Weapon type and count
> Present location (latitude, longitude, altitude)
> Present speed
> Present heading
> Weapon launch location.

b. Threats—Threat models describe the characteristics of cruise missiles, drones, or other munitions released from threat platforms that are intended to destroy defended assets. Threats will have a defined target or targets and in the case of nuclear threats, may be salvage fused. Threat characteristics that define the behavior of a threat are as follows:

> Threat type (ALCM, SLCM, High-Fast, Low-Slow, etc.)
> Warhead size/type (nuclear, chemical, conventional)
> Way points
> Speed
> Location (latitude, longitude, altitude)
> Heading
> Target location/detonation altitude (latitude, longitude, altitude).

c. C2 Platforms—A C2 Platform is any airborne or surface element which acts as a fusion point for sensor data and/or the tasking of weapons. The C2 platform executes C2 node models. The individual C2 platform models are based on the specific platform type and the following characteristics, as applicable:

> Platform type
> Flight profile/movement characteristics
> Present location (latitude, longitude, altitude)
> Present speed
> Present heading
> Loiter/orbit time remaining
> Communication links (Tx/Rx, number and type, primary or secondary).

d. Weapon Platforms—Weapon platforms are weapon carriers, such as interceptors and SAMs, which have the capability to neutralize a threat. Weapon platforms are directed by C2 nodes to engage targets. The weapon platform is responsible for verifying the intercept/guidance solution from the C2 node. The weapon platform's target acquisition sensor is subject to sensor detection processing to determine if it can "detect" the threat. This "macro-C2" model results in detection, processing, and weapon release on a local scale for the weapon platform.

Weapon platform characteristics are (as applicable):

> Weapon platform type
> Weapon type/count
> Location (latitude, longitude, altitude)
> Heading
> Speed
> Flight profile/movement characteristics
> Loiter/orbit time remaining
> Communication links (Tx/Rx, number and type, primary or secondary).

Additionally, weapon platforms communicate with a C2 node to provide the information listed above, as well as the following:

> Weapon availability/status (out of action/firing, etc.)
> Target engagement status (tracking, lost track, etc.)
> Weapon effectiveness (kill, no kill).

The weapon platform's target acquisition sensor is subject to sensor detection processing to determine if it can "detect" the threat. This "macro-C2" model results in detection, processing, and weapon release on a local scale for the weapon platform.

e. Weapons—Weapon models are used to determine weapon effectiveness against threats and threat platforms. Weapons are launched from weapon platforms to destroy threats and threat platforms.

The weapon model uses the following weapon characteristics to determine weapon effectiveness (i.e., probability of kill (Pk)):

> Weapon type
> Target type
> Launch location
> Range to target
> Speed
> Intercept geometry
> Environmental effects.

f. Sensor Platform—Sensor platforms can be fixed or mobile and carry one or more sensor types for detecting and classifying objects. Sensor platforms communicate with C2 nodes to report object detections. Object detections are determined via the sensor detection model. Sensor platforms have the following characteristics (as applicable):

> Sensor types
> Location (latitude, longitude, altitude)
> Speed
> Heading
> Flight profile/movement characteristics
> Loiter/orbit time remaining
> Communication links (Tx/Rx, number and type, primary or secondary).

g. Neutral Platform—Neutral platform models are utilized to generate/represent the object loading on C2 sensor and weapon models as a result of normal civilian, and tanker, air traffic. Flights within the continental boundaries and outbound transoceanic tracks are represented as a statistical distribution for object loading. This statistical distribution will not be uniform but will be geographically sensitive based on FAA flight information. Transcontinental flights entering the ADIZ are modeled as individual platforms with the following parameters:

> Location (latitude, longitude, altitude)
> Speed
> Heading
> Destination
> Way point/flight path
> IFF/SIF modes and codes.

The transoceanic inbound neutral models are used to simulate potential masking of threat platforms and threats due to minimal object separation.

10. Sensor Processing Models

The sensor environment model determines the cells into which the sensor has the capability to detect targets and the decrease in Pd due to environmental conditions and enhanced environment effects. This model provides an input to the sensor detection model from which the Pd for each target is computed. Moving sensor platforms such as the ASTS are a special case of this mode. Moving platforms have their cell coverage area redefined on a periodic basis due to their location changes and resulting change in their sensor coverage area. The decrease in Pd value is computed and stored for each grid cell by sensor type and location.

11. Sensor Detection

Sensors are active or passive (e.g., radars, infrared detectors, and electro_optical (EO)) devices for detecting objects. The detection performance characteristics for each sensor are defined in terms of parameters related to each sensor type (radar cross-section, range, IR signature, etc.). For example, radar cross-sections in three dimensions as a function of aspect angle will be computed in the sensor model for defender platforms, weapons, threat platforms, and threats. These values will determine the Pd for each object and subsequently will give rise to the generation of messages to the associated command element.

a. Sensor Processing—The sensors use grid cells to determine coverage area and Pd. Each object in every cell within coverage range is checked for detection probability.

Object detection criteria are sensitive to range, aspect angle, 3D crosssections, terrain masking, and environmental effects.

b. Sensor Message Generation—The sensors generate messages for transmission to their associated command element for detected targets. The message formats and characteristics are determined by the originating sensor. The messages should contain at least the following information:

> Reporting Sensor ID.
> Destination ID.
> Time Tag—detection time of the target.
> Target Location—represented in geographic coordinates (latitude/longitude), or
>> other similar surface identification)
> Target Altitude (if available for the sensor).
> Target Speed (if available for the sensor).
> Target Heading (if available for the sensor).
> Sensor Location—location at time of detection.
> IFF Modes and Codes (if available for the sensor).
> Probable Target Type and Confidence Level (if available).

Sensor message types are (as applicable):

>Target Report—Radar, IR and EO type sensors.
>Strobe Report—azimuth only return due to ECM and ECCM effects on radar type sensors.
>Status Report—highest priority message, transmitted periodically or at the start of scan; determines operability of sensor.
>IFF/SIF Report—IFF radar report with modes and codes.
>Track Report—target report message from sensor with integral C2 facility such as E-3 or their C2 facility where target reports have been pre-processed.

12. Communication Link Models

Communication links provide the path and the medium through which message traffic to/from sensor, weapons, and C2 platforms flow. Communication link model parameters are described below.

a. Link Type. The communication link type defines the means of communication between the transmitting and receiving nodes. The link types modeled are:

>Radio and band
>Landline
>*Laser communication*
>Microwave
>Fiber optic
>Network.

The link types are associated with unique rule sets to determine their behavior under normal (clear transmission) conditions as well as degradation due to environmental effects.

b. Link Connectivity. The communication link model verifies connectivity between the transmitter and receiver based on the source and destination information contained in the message. Connectivity checks are based on the following:

>Availability of transmitter and receiver
>Path availability (i.e., LOS, link media exists).

c. Link Capacity. The link type will define the link capacity in terms of maximum messages transmittable per time period. The model verifies the buffer status (full or empty) and the link transmission rate to determine throughput capability.

III-52

d. Link Degradation. Communication links are affected by various environmental effects depending upon link type. The link degradation will be computed by taking into account the environmental conditions for all cells which the signal must traverse (radio, microwave, and laser signals only).

13. Command and Control (C2) Node Models

The C2 decisions and processes (DAP) activity and C2 databases interact to simulate the C2 node functions. The C2 DAP function consists of programmed decision logic which is defined in terms of rules of engagement which generate messages for transmission to other platforms for the purpose of neutralizing the threat. Every C2 node will contain a C2 DAP model with which the C2 database interacts and which causes the generation of messages.

a. C2 Decisions and Processes. The C2 decisions and processes is responsible for sensor input, target identification, sensor data correlation and tracking, air situation display, and message generation for purpose of engaging threats and forwarding air situation information to other C2 nodes as defined by the C2 platform communication link connectivity list.

b. C2 Received Messages. The C2 decision and processes function receives the following message from external sources:

Track Messages (from other C2 nodes)
Target Report Messages
Strobe (jammer) Report Messages
Sensor Status Messages
SDI Model Messages
Asset Status Messages
Weapon Status Messages
Engagement Reports from Weapons
Engagement Reports from Other C2 Nodes
Weather Information Messages
Platform Location Messages
C2 Status Messages from Other C2 Nodes
Weapon Effectiveness Messages
ASW Model Messages.

c. C2 Transmitted Messages. The C2 DAP function transmits the following messages to external sources:

Track Report Messages to other C2 Nodes
Scramble/Intercept Messages
Engagement Messages
C2 Location (platform) Report Message to other C2 Nodes
C2 Status Message (percent saturation, remaining assets) to other C2 Nodes.

d. C2 Decisions and Processes Rule Set. The C2 DAP function are responsible for the following functions. The decisions are based on C2 database interaction, rules of engagement, weapon effectiveness, and available resources.

> ID Determination/Validation
> Threat Assignment
> Weapons Allocation
> Resource Protection Prioritization
> Threat Type
> Tracking/Correlation.

14. C2 Database

The C2 database is responsible for maintaining an accurate and current accounting of all threats, defender assets, and environmental conditions applicable to the simulation. The C2 database must also interact with the C2 DAP function. A C2 database containing an appropriate subset of the following information is maintained at each C2 node.

a. Site data for this C2 Platform (as applicable for fixed or mobile platforms): Grid Position, Speed, Heading.

b. Communication Link List for this Platform Type: Number of Landlines; Medium: Fiber Optic, HF, UHF, VHF; Characteristics Tx or Rx Destination (if Tx) and/or Source (if Rx) Data Rate.

c. Threat Platform/Threat List: Grid Position, Speed, Heading, Engagement Priority, Weapon Commitment/Assignment, Type, Confidence of Type, Assets at Risk Confidence, IFF, Queries/Responses, Threat Summary.

d. Weapon Platform/Weapon List: Grid Position, Heading, Speed, Max Speed, Fuel Burn Rate, Max Altitude, Max Range, Flight Profile, Weapons Available/Type Number, AAM/ASM, Bullets, Alert Status (delay to deployment), Fuel/Time Remaining, Target Assignment, CAP/Orbit Time/Location of Intercept, SAM Batteries, Missile Count Status.

e. Asset Summary Data, C2 Centers, Airbases, Aircraft/Type, SAM Battalions Batteries/Missile Count, Space-Based Platforms, Sensors, Industrial Centers, Population Centers, Other Defended Assets, Defense Priority for Each Asset.

f. Sensor Platform List (as applicable for fixed or mobile platforms): Grid Position, Heading, Speed. Sensor Type, SR, SSR, 3D Radar, ELINT, Space-Based IR/Radar, EO, IR Data Count, Number of Targets, Types (SR, SSR, etc.) Data, Priority/Confidence Level.

g. Other/Commercial Traffic: Grid Position, Heading, Speed, ID tanker, friendly, commercial. unknown, etc.), ID Confidence Level.

h. Nudet Reports

i. Weather

j. Enemy Order of Battle.

15. SDI Models

SDI inputs are pre-formatted messages transmitted to the destination defined by the message at predetermined times in the simulation. The message may be of the following types: slow walker, missile attack warning, nuclear detonation, asset warning, and situation assessment messages.

16. ASW Models

ASW inputs are preformatted message transmitted to the destination defined by the message at predetermined times in the simulation. The messages may be of the following types: SLCM launch or submarine location.

3.4.3 Processing Parametrics

To define and refine suitable configurations for the TEF, parametric limits were established. These limits encompassed the following areas:

> Object and model requirements
> Data base characterization and sizing
> Pre-test inputs and sizing
> Model loading, data extraction and runtime requirements
> Post-test outputs and response turnaround time.

A scenario was postulated to provide processing load numbers and durations. The goal was to have a realistic attack scenario which stressed all models. The study indicated that there were two basic types of scenarios, a leading-edge cruise missile attack or a follow-on attack after a ballistic missile attack. Various levels of warning and readiness could be assumed for either type. The chosen scenario represents an eleven-hour duration leading-edge cruise missile attack due to its stressing nature.

3.4.4 Load Analysis

An estimate of the total simulation executable instruction throughput requirements was performed to support hardware and software architecture and configuration analysis and selection. The models previously described were used to represent the air defense system architecture and threats. A representative air defense architecture based on evolution from the current JSS system to an

advanced system was used as a baseline. The threat loading for the air defense architecture was determined based on a postulated leading-edge cruise missile attack scenario. The architecture and attack scenario were chosen to represent the most stressful load on the TEF. The attack scenario was divided into eleven one-hour time intervals.

3.4.4.1 Interactions Among the Models

Model interactions (i.e., the numbers of aircraft detected and reported by sensors), were assumed to remain constant throughout each time interval. The number of instructions, frequency of computation (period) and the total number of object interactions were estimated for each model for all time slices. The number of instructions executed on a per model basis as well as total instructions for the entire time slice were computed. A spreadsheet was used to store the data on the models and interactions and to calculate the totals (attached). The summary of results from two separate loading estimates is shown in Tables 3-9 and 3-10 (following pages). These tables show the models, time slices, instruction count estimate and the percentage of instructions per time slice, and the total instruction count estimate and the percentages over all time slices.

Table 3-9 shows a large proportion of the entire simulation processing devoted to SOCC processing. Close examination of these numbers revealed a bias due to an assumption that all SOCCs would have redundant processing for all functions. This assumption tripled the SOCC basic instruction count.

3.4.4.2 Sizing Analysis

This high SOCC load and resulting high total instruction count lead to the conclusion that a simulation with an average turnaround time of 10 to 18 minutes might not be achievable within the limits of current technology. In Table 3-10, the redundant SOCC processing was eliminated. The total instruction count was accordingly reduced, and it was determined that a simulation of this order of magnitude was achievable. The first estimate (with redundant SOCC processing) required a total compute power in excess of 67 MIPs, whereas the second resulted in an estimated compute power of 30 MIPs. System architectures based on redundancy of processing in SOCCs or other elements that require very large processing times could be run, but probably could not achieve the desired turnaround time of 10 to 18 minutes.

3.4.4.3 Storage Estimates

The memory required for hosting and executing a simulation was derived as follows. The memory required to hold the data arrays from the scenario file is shown in the following table. The output buffers for data extraction were estimated at 250 KBytes and the operating system was allocated 1000 KBytes of memory. The total required for a simulation is 35 MBytes.

Table 3-9. Initial TEF Load Analysis

TIME SLICE	1		2		3		4		5		6		7		8		9		10		11		TOTALS	
MODEL	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%
OBJECT UPDATE	3.5E7	3	3.6E7	2	4.0E7	1	4.4E7	1	3.6E7	1	3.3E7	1	2.9E7	1	3.5E7	1	2.5E7	1	3.1E7	1	8.5E6	1	3.7E8	1
ENVIRONMENT	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.0E4	·	4.4E5	·
NUCLEAR	0	·	0	·	4.9E5	·	7.0E5	·	1.3E6	·	2.1E6	·	3.5E6	·	6.6E6	·	1.2E7	·	1.8E7	·	3.0E7	·	7.6E7	·
SENSOR UPDATE	1.8E7	1	1.8E7	1	1.8E7	·	1.8E7	·	1.8E7	·	1.8E7	·	1.8E7	1	1.8E7	·	1.8E7	·	1.8E7	·	1.8E7	1	2.0E8	1
COMM LINKS	2.0E5	·	3.7E5	·	1.6E6	·	1.6E6	·	1.6E6	·	1.6E6	·	1.4E6	·	1.4E6	·	1.2E6	·	1.2E6	·	1.1E6	·	1.3E7	·
SENSOR MODELS	1.4E8	9	1.4E8	9	1.4E8	3	1.1E8	2	8.7E7	2	7.0E7	2	6.8E7	2	1.1E8	3	7.1E7	2	7.0E7	2	2.0E7	2	1.0E9	3
COMM	4.9E8	33	5.0E8	31	3.9E8	8	2.9E8	6	2.4E8	5	1.7E8	4	1.7E8	4	2.3E8	6	1.7E8	5	1.9E8	5	7.0E7	3	2.9E9	7
SOCC	7.6E8	52	9.1E8	57	4.0E9	80	3.9E9	81	3.8E9	81	3.7E9	84	3.1E9	82	3.7E9	80	2.7E9	82	2.7E9	79	2.1E9	81	3.7E10	80
RDCC	1.0E6	·	1.9E6	·	2.7E6	·	2.1E6	·	2.0E6	·	1.9E6	·	2.2E6	·	3.7E6	·	3.0E6	·	1.9E6	·	1.0E6	·	2.4E7	·
CMC	1.0E6	·	1.9E6	·	1.9E6	·	1.9E6	·	1.9E6	·	1.9E6	·	1.9E6	·	1.9E6	·	1.9E6	·	1.9E6	·	1.9E6	·	2.0E7	·
INT	2.1E8	·	2.9E7	2	4.4E8	9	4.4E8	9	4.4E8	9	4.0E8	9	4.0E8	11	4.0E8	10	3.3E8	10	3.3E8	10	2.7E8	10	3.5E9	8
SAM	→		→		→		→		→		NEGLIGIBLE													
DATA RECORD	→		→		→		→		→		NEGLIGIBLE													
TOTAL	1.5E9	4	1.6E9	4	5.0E9	13	4.8E9	12	4.7E9	12	4.4E9		3.8E9	10	4.0E9	10	3.3E9	10	3.4E9	8	2.6E9	7	4.0E10	·

III-57

Table 3-10. TEF Load Analysis Without Redundant SOCC Processing

TIME SLICE	1		2		3		4		5		6		7		8		9		10		11		TOTALS	
MODEL	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%	COUNT	%
OBJECT UPDATE	3.9E7	4	3.9E7	4	4.8E7	2	4.4E7	2	3.8E7	2	3.3E7	2	2.9E7	2	3.5E7	2	2.5E7	2	1.2E7	1	8.5E6	1	3.5E8	2
ENVIRONMENT	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.0E4	-	4.4E5	-
NUCLEAR	0	-	0	-	4.9E5	-	7.0E7	-	1.3E6	-	2.1E6	-	3.5E6	-	8.6E6	-	1.2E7	-	1.8E7	1	3.0E7	3	7.6E7	-
SENSOR UPDATE	1.8E7	2	1.8E7	2	1.8E7	1	1.8E7	1	1.8E7	1	1.8E7	1	1.8E7	1	1.8E7	1	1.8E7	1	1.8E7	1	1.8E7	2	2.0E8	1
COMM LINKS	2.0E5	-	3.7E5	-	1.6E6	-	1.6E6	-	1.6E6	-	1.6E6	-	1.4E6	-	1.4E6	-	1.2E6	-	-1.2E6	-	1.1E6	-	1.3E7	-
SENSOR MODELS	1.4E8	14	1.4E8	13	1.4E8	6	1.1E8	5	8.7E7	4	7.0E7	4	8.0E7	4	1.1E8	8	7.1E7	5	3.9E7	3	2.0E7	2	9.9E8	6
COMM	4.9E8	51	5.0E8	45	3.9E8	18	2.9E9	13	2.4E8	11	1.7E8	9	1.6E8	9	2.3E8	12	1.7E8	11	1.0E8	7	7.6E7	6	2.9E9	15
SOCC	2.6E8	27	3.0E8	27	1.3E9	54	1.3E9	59	1.3E9	62	1.2E9	60	1.0E9	59	1.1E9	58	9.0E8	60	9.0E8	61	7.1E8	59	1.0E10	56
ROCC	1.9E6	-	1.9E6	-	2.2E6	-	2.1E6	-	2.0E6	-	1.9E6	-	2.2E6	-	3.7E6	-	3.0E6	-	1.9E6	-	1.0E6	-	2.4E7	-
CMC	1.8E6	-	1.8E6	1	1.8E6	-	1.8E6	-	1.8E6	-	1.8E6	-	1.8E6	-	1.8E6	-	1.8E6	-	1.8E6	-	1.8E6	-	2.0E7	-
INT	2.1E6	-	2.9E7	3	4.4E8	18	4.4E8	20	4.4E8	21	4.0E8	20	4.0E8	24	4.0E8	21	3.3E8	22	3.3E8	24	2.7E8	23	3.5E9	19
SAM	0	-	0	-	0	-	8.7E5	-	8.0E5	-	8.0E4	-	0	-	1.1E8	-	8.7E6	-	8.8E6	-	8.3E4	-	1.1E7	-
DATA RECORD	2.0E7	2	2.0E7	2	1.7E7	1	1.5E7	1	1.4E7	1	1.3E7	1	1.3E7	1	1.4E7	1	1.4E7	1	1.3E7	1	1.4E7	1	1.7E8	1
TOTAL	9.7E8	5	1.1E9	6	2.4E9	13	2.2E9	12	2.1E9	12	2.0E9	11	1.7E9	9	1.9E9	11	1.5E9	8	1.4E9	8	1.2E9	7	1.8E10	-

1. Sizing Analysis

Object, Environment, and Model Definition	3,585 Kbytes
Terrain Data	140 Kbytes
Input Buffers	40 Kbytes
Subtotal	3,765 Kbytes

The major size estimates of the data structures are:

Event Queue	80 Kbytes
Object Truth	1,200 Kbytes
Sensor Files	10,000 Kbytes
Sensor Message Buffers	3,000 Kbytes
C2 Track Files	12,000 Kbytes
Weapon Track/Detection Files	1,200 Kbytes
C2/Weapon Message Buffers	1,200 Kbytes
C2/C2 Message Buffers	920 Kbytes
Subtotal	29,400 Kbytes
Total	33,165 Kbytes.

3.4.5 Database Approach

Given the size of the experiment, the number of experiments to be run, and the responsiveness needed, the TEF must have an efficient means to build, access, and control the data. The key to providing this is a relational database for object definition, model specification, scenario definition, and post-test data storage. In addition to the relational database and its associated Data Base Management System (DBMS), three other data structures are key to database sizing; these key data structures are the actual scenario files needed for test execution, the data extraction files produced by test execution, and the application source code for the TEF simulation.

3.4.5.1 Relational Database

The approach for estimating the size of the DBMS is based on the following:

a. The categories and number of types of objects provided the basis for estimating the number of records and some of the elements within each record for the objects.

b. The model descriptions expanded the elements required for some objects and provided the basis for estimating the number of records and the number of elements per record for the models.

c. The MOE/MOP definitions and the experimental plan enumerations provided the basis for estimating the number of records and the number of elements per record for post-test data storage.

3.4.5.2 Object Storage Sizing

The objects are subdivided by group, category, kind, and type. There is an average of 50 parameters associated with each type. Each parameter requires approximately 4 bytes of storage and would be an element within a record. The number of elements per record is equal to 50. Definition of each element requires an overhead of 120 bytes per unique element. Definition of a record requires an overhead of 1400 bytes for each unique record type. Using these assumptions the estimates for Object Storage in this DBMS were derived.

Object Storage Sizing

Number of Unique Records	60
Overhead Per Record	x 1,400
Subtotal	84,000 bytes
Number of Unique Elements per Record	50
Overhead per Element	x 120
Unique Records	x 60
Subtotal	360,000 bytes
Number of Elements per Record	50
Total Number of Records	x 2,350
Number of Bytes per Element	x 4
Subtotal	470,000 bytes
Total Number of Records	2,350
Key Size in Bytes	x 96
Subtotal	225,000 bytes
Total	1,139,000 bytes

3.4.5.3 Model Parameter Storage Sizing

There are ten different categories of models; Environment, Nuclear Effects, Object Motion, Sensor Environment, Sensor Detection, Communication Links, C2 Nodes, Weapons, SDI Input, ASW Input. The number and size of the parameters associated with each of the models was estimated and then multiplied by the number of possible types. The total storage requirement DBMS is 80 Mbytes.

1. Environment—Atmospheric environment parameters storage:

> 40 bytes/type x 80 types of environmental conditions = 3200 bytes.

> 75 degrees latitude x 360 degrees longitude = 27,000 one degree cells.

> 3200 bytes x 27,000 cells x 6 scenarios = 518.4 Mbytes of data storage.
>   (This data is not held in the DBMS).

The terrain is modeled by a four-minute-by-four-minute grid with two parameters per cell.

The North American continent reaches from 15 00 00 North Latitude to 75 00 00 North Latitude, and from 45 00 00 West Longitude to 180 00 00 West Longitude.

> (60 deg x 15 cells/deg) x (135 deg x 15 cells/deg) = 1.82 Mbytes

> 1.82 Mbytes x 8 bytes = 14.56 Mbytes.

(Neither this data nor the Enhanced environment or Electromagnetic effects is held in the DBMS).

2. Nuclear Effects—An estimated array of 100 parameters per warhead type is necessary to provide burst and fireball data parameters to the online model. This estimate is based on past Logicon experience with similar simulations.

> 20 warhead types x 4000 bytes = 80,000 bytes

3. Object Motion—Data for defining movement patterns, formations and missions for all mobile categories is computed as:

> 64 bytes/event x 20 events/patterns x 10 patterns/object type x 150 object types
>   = 1.9 Mbytes

4. Sensor Environment—An estimated array of 100 parameters for defining the cell coverage, noise sensitivity, and environment sensitivity for each type of sensor is necessary.

> 400 bytes x 50 sensor types = 20,000 bytes

5. Sensor Detection—The primary data requirement is to hold the cross-section per object type per sensor type.

20 bytes/cross section x 120 objects x 50 sensors = 12,000 bytes

There are 100 other detection and reporting parameters per sensor type.

400 bytes x 50 sensor types = 20,000 bytes

6. Communication Links—The main requirement is the connectivity definition and the rules that determine the behavior of each type of link.

40 bytes/connection x 1677 links
= 67,080 bytes 100 bytes x 40 link types
= 4,000 bytes

7. C2 Nodes—The key data parameters for the C2 models are the decision logic parameters for track generation, target ID, threat ranking, weapon allocation, and kill assessment. For each C2 type:

80 bytes/rule x 500 rules = 40,000 bytes per C2 type.

Asset, sensor, communication, and other definition data requires about 36,000 bytes per node. The total data storage would be 1.8 Mbytes.

8. Weapons—The weapons effectiveness and engageability definition data would require about 74 Mbytes.

9. SDI Input—This input consists of an ordered set of events. The time is relative to the initiation of the ballistic missile attack. This data will not be held in the DBMS.

10. ASW Input—The data for this model consists of the detection zones and report generation time cycles and delays for the ASW Model. The data size is estimated to be 40,000 bytes.

3.4.5.4 Post-Test Data Storage

The data to be held in the DBMS consists of the computed MOE/MOPs from either run or the set of runs that provides statistical significance for each experiment/case combination. This data is estimated to be about 2000 bytes per iteration with from 1000 to 25000 iterations per year, the data saved in the DBMS must be kept to a minimum.

3.4.5.5 Scenario File Storage Sizing

The estimated scenario file storage size is estimated as follows:

Object Definition	12 Kbytes
Sensor Definition	80 Kbytes
COMM Link Definition	72 Kbytes
Environment Definition	67 Kbytes
C2 Node Definition	1,800 Kbytes
Terrain	14,560 Kbytes
Weapon	15,000 Kbytes
SDI Input File	19 Kbytes
ASW Definition	6 Kbytes
Object Control File	11,520 Kbytes
Location Definition	48 Kbytes
	===============
Total Size per Scenario	43,184 Kbytes

3.4.5.6 Data Extraction Storage Sizing

The estimate was based on extraction of the C2 mode data and object position once per minute during test execution for total of 40 executions. The amount of the data extracted was estimated to be 1 Mbyte per extraction; this yields a total of 40 Mbytes.

In addition, the counts used for generating MOEs will be extracted on the appropriate cycle update. This would amount to approximately 15 Mbytes. The total amount of data produced per run would be 55 Mbytes. This could be reduced, through data reduction techniques, to as little as 21 Kbytes per run. This data in turn could be further reduced to one set of data for the number of runs required to produce statistical significance.

Only the mean, standard deviation, variance, minimum, and maximum values for each MOE are shared from the set of runs. This is estimated to require about 2 Kbytes for the data. Approximately 1,080 sets of data per year would be stored online in the DBMS for quick retrieval and comparison. This would result in a minimum 2.16 Mbytes of data per year being added to the DBMS due to post-test generation of test results.

3.4.6 Parallelization of Simulation Functions

Each of the individual model modules described above that results in object motion (weapons, neutral, and threat), sensor detection or environmental calculations are subject to parallelization. The calculations performed are identical for all objects of the same category and parallelization offers the

potential for greatly increasing efficiency by performing parallel calculations on non-interactive items. In the case of the executive, however feasible parallelization might be, the distribution of functions must be very carefully weighed in the light of the necessity for simulation repeatability. The consequence of such analysis might lead to the decision that it would be better to keep it executing in a non-parallel fashion. The C2 model is likely to be represented by a rule-based process.

In each of the cases, non-von Neumann architecture solutions offer the possibility of achieving a greater degree of efficiency than nonparallel architectures. However, the system should be designed specifically for non-von Neumann architectures in order to realize the improved efficiency.

3.4.7 Candidate Host Computer Configurations

Three different NvN architectures were considered;

> Class I: Pipelined Vectorized Uni-Processor
> Class VI: General Purpose, Multiple-PEs, Shared Memory
> Class VI: General Purpose, Multiple-PEs, Message Passing

3.4.7.1 Class I: Pipelined Vectorized Uni-Processor

At the present time, the only computers with the single instruction stream that meet or exceed the 40 MIPS requirement are the true supercomputers as represented by CRAY as well as several other multi-million dollar computers. Even these machines achieve this throughput only on highly vectorized problems. Multiflow Computers reports that it is close to releasing its TRACE 28/200 which, they claim, will achieve these performance levels for conventional non-vectorized FOR-TRAN programs. Multiflow Computers claims that users can expect performance in excess of 50 MIPS. The TRACE 28/200 can be created with a field upgrade of hardware and software from their current product TRACE 7/200; which, itself, delivers more than 15 MIPS.

A simulation executive based on a uni-processor hardware foundation is considerably simpler to develop than one based on Class VI NvN architecture because there is no necessity for complex and complicated synchronization of parallel processes.

3.4.7.2 Class VI: General Purpose, Multiple-PEs/Shared Memory

There is a wide variety of shared memory GPMPE computers available. The marketplace offerings include: Alliant, Flexible, Sequent, Encore, and Elxsi.

Individual processor performance should exceed 12 MIPS and 1 MFLOP as measured by the half-precision LINPACK benchmark. Of the currently fielded products, only Alliant, Multiflow, and Elxsi are able to achieve this performance level.

Encore, among others, claims to have upward compatible products that can achieve these perform-
ance levels. To indicate future possibilities, Encore is under contract to DARPA to deliver a 1 BIP
ma.. ine complex, which will contain several 20-CPU Encore machines that will be cross-coupled
using high-bandwidth fiber optic channels.

The Alliant FX series is a currently available, shared memory machine. The FX-40 and FX-80 series
provide a unique architecture of up to eight Compute Elements (CEs) and up to 12 Interactive
Processors. Each CE is capable of approximately 14 MIPS (Whetstone rating) and 2 MFLOPS (half-
precision LINPACK benchmark). Each CE has vector instruction capability. Moreover, CEs can
be ganged together to constitute a computing complex. This mode is supported by a unique compiler
which automaticallly generates "fingergrained" (medium grained) parallel code as well as vectorized
code. A specially patented hardware bus, that Alliant calls a concurrency bus, provides the essential
microsecond level processor synchronization. It is possible to realize 96 MIPS with a single Alliant
FX-80.

The Elxsi 6400 series is also a GPMPE-Shared Memory machine. The Elxsi architecure offers the
possibility of 10 CPU boards which operate in arbitration via a high-speed gigabus using a technique
Elxsi refers to as dynamic load balancing. In other words, the Elxsi operating system decides how
many CPUs to use when a job is executed. Each 6460 CPU board provides 40 MWHET performance
and is rated at 7 MFLOPS. Elxsi also claims "special" capability to accommodate real-time and
simulation software systems.

3.4.7.3 Class VI: General Purpose, Multiple-PEs/Message Passing

The US Army's Conceptual Modeling Agency (Bethesda, MD) has stated to CSC's High Perform-
ance Computing Laboratory personnel that although it is easy to assume that the message passing
subclass of the NvNACS Class VI machine is not as suitable for the compute-intensive processes
involved in real-time simulation of BM/C3I systems as the shared memory subclass machines, their
experiences do not substantiate that assumption. They admit that it requires a different kind of
programming, but the necessary capabilities for such c.mulation are indeed present.

3.5 SIGNAL PROCESSING APPLICATIONS OF NVN ARCHITECTURES

3.5.1 Signal Processing Generic Definition

Signal processing is the application of algorithms to sampled data from single or multiple sensors for
the purpose of extracting intelligence from the data and/or improving the quality of intelligence that
may be extracted.

Signal processing techniques are applied to many types of signals including:

> telecommunication signals (both voice and data)
> radar signals

video images (infrared sensors)
acoustic signals (e.g., sonar, speech, music)
seismic signals
medical instrumentation signals (e.g., EKG, ultrasound)
accelerometer data

The processing algorithms are applied for a variety of purposes, such as:

improvement of signal to noise ratio
speech recognition/speech compression
detection of events (e.g., target search)
pattern recognition (discrimination between events)
parameter measurement (location, speed, source energy, spectral analysis)
target tracking and surveillance
beamforming (acoustic, radar, seismic)
image processing (medical)
vibration analysis

3.5.2 Signal Processing Problems

The most pervasive problem of signal processing is its computational intensity. In some cases relatively high I/O bandwidths are also required, but computational bandwidth is the predominant problem.

The problem of high data rates from a large number of sensors is aggravated by the additional requirement for high precision computation when using the more sophisticated processing algorithms. Advances in signal processing over the past three decades have brought increasing complexity of the algorithms, ranging from filtering to spectral analysis to adaptive beamforming. These changes in algorithmic complexity have altered the computational load from a factor of N to a factor of N^2 to a factor of N^3 (where N is the number of data samples to be processed in a given time period). In most signal processing applications, the processing load must be handled in real-time.

A common and significant attribute of most signal processing applications is the use of complex mathematical techniques such as FFT (fast Fourier transform), IIR (infinite impulse response) filtering, FIR (finite impulse response) filtering, and matrix operations. This algorithmic commonality makes it feasible in many instances to select or to design a system architecture that is suitable for multiple signal processing applications.

Applying NvN architectures, i.e., multiple parallel or pipelined processors, to signal processing is not new. In fact, the requirements for signal processing have been a significant driving force in the development of NvN architectures. However, the system engineering question of which architecture(s)/ hardware are best suited to particular signal processing applications has an equally important

companion question of what the software development implications are of using a specific architecture for a particular application.

Having selected an architecture that is characterized by extensive parallelism and/or pipelining, there remain: the problem of how to make the most effective use of the hardware capabilities through software. The major steps to be taken toward solving this problem are:

1. Develop operating system(s) that simplify the application program interface to the basic capabilities of the hardware.

2. Develop a high order language, or new constructs for an existing language, that allow application programmers to use the architectural features simply and straight forwardly.

3. Develop intelligent compilers/code configurers that have enough knowledge of the hardware to be able to recognize potential opportunities to utilize the architectural features. [This has been indicated as a key tool for architecture classes VI.1 and VI.2]

4. Develop libraries of programs and program segments that make use of the hardware's architectural features; this supports re-use of existing code and reduces the cost of application development.

5. Develop intelligent debugging tools, having significant knowledge of the underlying hardware and that can simplify and speed up the process of getting programs into regular usage.

3.5.3 Use of NvN Architectures in Signal Processing

Non-von Neumann architectures are already in use in most of the signal processing applications where computational bandwidth requirements indicate the need and where cost allows. Numerous pipelined array processors (not to be confused with processor arrays) of the class I type have been commercially available as peripherals to mainframe computers, and have been applied to many signal processing applications since the early 1970s.

Adaptive beamforming in radar, sonar, and seismic applications has been performed using rhythmic cellular architectures as well as processor array type architectures. Target tracking applications have also been performed on associative processor architectures. Processor arrays have also been applied to speech and image processing. Various multiple processing element (PE) architectures have been applied to general signal processing, including the application of expert systems technology to signal analysis.

3.5.4 Future Use of NvN Architectures in Signal Processing

The use of neural networks for existing signal processing applications probably will not increase dramatically in the near future; however, there is some probability that this architecture might give rise to totally new signal processing applications over the next decade.

3.5.5 Matching NvN Architecture Classes and Signal Processing Problems

The common usage of certain mathematical techniques across a broad spectrum of signal processing applications having significant computational bandwidth requirements allows straightforward porting of many applications to several different architectures. Consequently, any recommendation of an architecture for a particular application should be based on a consideration of different applications that could be supported by the recommended system. It should be borne in mind that, in any given instance, the total performance profile depends as much or more on software as on hardware.

3.5.5.1 Pipelined Vector Uniprocessors

Many application systems have been built on this architecture, both as general purpose computers and as special purpose signal processors. While this architecture does not offer the speed improvements offered by highly parallel architectures, it does yield significant improvements over traditional von Neumann architectures. In addition, they have the advantages of being lower in cost, and of being substantially less difficult to utilize efficiently than more highly parallel systems.

3.5.5.2 Rhythmic Cellular

Rhythmic cellular architecture (either systolic or wavefront) is the current preferred choice for real-time adaptive beamforming applications (radar, sonar, and seismic). This architecture is well suited to adaptive beamforming applications primarily because the data flow of the process maps directly into the architecture of the processor, allowing minimal overhead for data movement. Data enters the processor array only at its periphery. Generally, this direct mapping is an advantage, but it requires significant effort to partition the processing when the problem size exceeds the processor array size. This architecture has been used also for other applications similar to signal processing, such as interference cancellation, model fitting, linear algebra, and DNA sequence comparison. The primary limitation of this architecture is that, although very efficient for the particular application for which a given array was designed, there is very little flexibility in adapting the system to another application.

3.5.5.3 Processor Array

This architecture offers advantages of increased performance by a factor up to the size of the array, but this can be reduced by the overhead of communicating data between processors. In addition, like

the rhythmic cellualar architecture, it is difficult to partition a problem when the problem size is mismatched with the hardware array size. This architecture has been very effectively applied to applications of beamforming (radar, sonar, seismic) and of speech recognition.

3.5.5.4 Associative Processor

There has been only limited use of this architecture; it has been applied primarily to target tracking and surveillance subsets of signal processing applications. Additional applications may evolve, however, in areas characterized by significant database searching as well as signal processing functionality as, for example, in pattern recognition or in expert system support of command decision making.

3.5.5.5 Operand Driven

Search of the technical literature did not reveal any applications of this architecture to signal processing.

3.5.5.6 GP Multiple PE

This rather broad class of architecures has been used extensively in most existing signal processing applications. Partitioning of the application processes to processors, whether automatically by the system or explicitly by the application, can be critical to effective use of the capabilities and features of the architecture. The extensive flexibility offered by instances of the majority of the subclasses is most useful in developing signal processing algorithms.

3.5.5.7 Neural Network

Search of the technical literature yielded no evidence that this architecture has been applied to signal processing. Such lack of evidence is probably due to the facts that it is both new and complicated, and that there are few hardware implementations. This architecture, conceivably, could be applied to sophisticated tracking/surveillance applications and to speech recognition and processing.

3.5.6 Signal Processing Applications vs. Hardware

Table 3-11 matches NvN hardware to signal processing applications; it also identifies the manufacturer or developer of the hardware. The primary source of systems included in this list was the survey data from Subtask 1. However, some systems not identified in the Subtask 1 survey data were found in the literature search and were added to the list. Where the table shows simply signal processing the literature indicates that the systems are being used for signal processing, but gave no more specific information. Where no application is listed, the literature gave no indication that the systems are being, or have been, used for signal processing; however, this should not be taken to preclude the possibility.

Table 3-11. Signal Processing Applications vs. Hardware Systems

SYSTEM	MANUFACTURER	CLASS	APPLICATION
Alliant FX/80	Alliant Computer Systems Corp.	VI	
Anza-Plus Neurocomputing Coprocessor System	Hecht-Nielsen Neurocomputer	VII	
ASPRO	Loral Systems Group	IV	aircraft/ship tracking
Balance 8000 Balance 21000	Sequent	VI	signal processing
Boltzman Machine		VII	paper machine
BSP	Burroughs Corp.	III	seismic signal processing
Butterfly	Bolt, Beranek and Newman	VI	
CDC Star-100	Control Data Corp.	I	
CEDAR	Univ. of Illinois	VI	
Celerity 6000	Celerity		
CHiP	Purdue Univ. Washington Univ.	VI	
Cm*	Carnegie-Mellon Univ.	VI	
CMOS VLSI Neural Network	AT&T Bell Labs	VII	
Connection Machine	Thinking Machines Corp.	VI	FFTs
Convex C-1 XL/XP	Convex Computer Corp.	VI	seismic signal processing
COSMIC CUBE	California Institute of Technology	VI	
Cray X-MP/4	Cray Research, Inc.	VI	
Cray-1	Cray Research, Inc.	I	
Cyber 205	Control Data Corp.	I	
Cyberplus	Control Data Corp.	VI	
DADO2	Columbia Univ.	VI	signal processing expert systems
DAP	Active Memory Technologies	III	
Data Driven Machine 1	Univ. of Utah	V	

Table 3-11. Signal Processing Applications vs. Hardware Systems (continued)

SYSTEM	MANUFACTURER	CLASS	APPLICATION
ELI	Yale Univ.	VI	paper machine
ELXSI System 6400	ELXSI	VI	
Encore Multimax	Encore Computer Corp.	VI	
ETA-10	ETA Systems, Inc.	VI	
FACOM VP-200 Vector Processing System	Fujitsu, Ltd.	I	
FGCS	Univ. of Tokyo		
FLEX/32 Multicomputer	Flexible Corporation	VI	
GaAs Systolic Array Beamforming Controller	RCA(GE)	II	
Galaxy (YH-1)	People's Republic of China	I	
HEP	Denelcor	VI	
Hitachi S-810	Hitachi	I	
IBM 3081	IBM		
IBM RP3	IBM	VI	
iPSC-VX	Intel	VI	
Matrix-1	Saxpy Computer Corp.	II	seismic/sonar/radar signal processing
MIT Data Flow Computer	MIT	V	
MPP	Loral Systems Group	III	image processing
MWAP	Johns Hopkins APL	II	signal processing
NEC SX-2	NEC	I	
NETL	Carnegie-Mellon Univ.	VII	
Neural Phonetic	Helsinki Univ. of Technology	VII	
Nobeyama FX	Fujitsu/Nobeyama Radio Observatory		radio astronomy/ spectroscopy
NON-VON(1/3)	Columbia Univ.	VI	

Table 3-11. Signal Processing Applications vs. Hardware Systems (continued)

SYSTEM	MANUFACTURER	CLASS	APPLICATION
Odeyssey	Texas Instruments		speech recognition EKG analysis
PASM	Purdue Univ.	VI	
P-NAC	Princeton Univ.	II	DNA sequence comparisin
PSC	Culler Scientific Systems	VI	
SCS-40	Scientif Computer Systems	VI	
SLAPP	Naval Ocean Systems Center	II	sonar adaptive beamforming
STARAN	Loral Systems Group	IV	
STC-RSRE wavefront array processor	Stabdard Telecommunications/ Royal Signals and Radar Est.	II	radar adaptive beamforming
Systeolic Adaptive Beamformer	ESL, Inc.	II	sonar adaptive beamforming
Systolic/Cellular System	Hughes Research Laboratories	II	linear algebra/ signal processing
T-ASP	Motorola (Canada)	III	real-time signal processing
Tagged Token Dataflow Machine	MIT	V	
TI-ASC	Texas Instruments	I	
TRAC	Univ. of Texas at Austin	VI	
Ultracomputer	New York Univ.	VI	
Vortex	Sky Computer		
WARP Machine	Carnegie Mellon Univ.	II	

3.6 IMAGE PROCESSING

Image processing has been defined in terms of two categories of processing by S.Y. Kung [Kung 1988]. From the text it is stated that "the research activities dealing with images are now divided into two disciplines: image processing and image analysis. Image processing consists of enhancement, restoration, reconstruction and coding, et c. Image analysis, on the other hand deals with extraction of lines, curves, and regions in images, classification of objects, texture analysis, analysis of moving objects, and scene analysis. Most image processing tasks are very time consuming. For example,

low level operations, such as filtering or enhancement, typically require the order of some tens of machine instructions per pixel. A typical image obtained from a LANDSAT earth resources satellite is about 1000 x 1000 pixels/image. This implies a computation requirement of some tens of millions of instructions per image, not including the computation for any substantive higher level processing. If such simple low level operations are to be performed at a video rate, say 25 to 30 frames per second this means a throughput requirement of about a billion instructions per second. In general, most real-time image processing throughput rates outstrip current parallel architectures. Thus image applications processing have long been (and will continue to be) a major driving force in the development of faster and more powerful parallel machines."

Image processing can be broken into two generic categories depending on the source of the data. These categories include the processing of satellite or aircraft digital downlinked imagery and the processing of photographic imagery onto a digital format satisfactory for later replay on a digital image processing system. Each image type requires the later generation of user products for evaluation of scene content/image enhancement. Due to the complexity of the processing of large volumes of digital imagery that are downlinked in raw data format from polar orbiters, this category of bulk image processing will be discussed in detail. Although polar orbiting Synthetic Aperture Radars will be available in the future and they could constitute a type of imaging platform that would be best amenable to the use of non-von Neuman architectures they have no proven record of daily collection of data in volume. (The first such instrument carried on SEASAT failed shortly after launch. Investigators are still evaluating the data acquired through the one-hundred days of life of SEASAT. Though these investigations have led to many useful and unique applications for the data they are limited by the limited quantity of data.) The only system that has a record of daily data collection and processing is the Landsat production system for Multispectral Scanner System and the Thematic Mapper system. MSS data have been made available to the user community since 1972. The TM was first carried on Landsat 4 launched in 1982.

3.6.1 Bulk Image Processing

Satellite image processing involves large volumes of data. Processsing systems for satellite imagery are required to perform high speed bulk processing of multispectral imagery data and includes a variety of functions for image enhancement, analysis, and classification. The Landsat Thematic Mapper instrument images seven spectral bands simultaneously using one hundred detectors on two focal planes. Four spectral bands of data in the visible and near infrared are acquired by silicon detectors on the primary focal plane. Each band is composed of sixteen detectors. Three spectral bands (data in the mid and far infrared) have their detectors mounted on a secondary focal plane that is cooled to assure the maximum signal-to-noise ratio for each detector. The thermal infrared band contains only four detectors. Image data are acquired through a combination of satellite motion along track and the oscillation of a primary mirror (7 hz) to gather cross track information. Data are multiplexed and packetized using the on-board computer interface prior to downlink of a bit serial data stream that must be modified for image processing. Landsat 4 TM was the first instrument built by NASA that merged the satellite pointing data, mirror scan profile, and other housekeeping data

with the video data stream. The coupling between the satellite motion, the mirror oscillation, and deviations from perfect satellite orbit trajectories requires that mathematical models be constructed to remove distortions induced by the total system.

Remote sensing technology continues to advance in its sophistication and technological difficulty in data processing. Two examples will be given of the future problems faced in these areas for the sake of illustration.

The first example concerns the use of linear or rectangular arrays of detectors. In this type of system the oscillating mirror is removed from the optical system. All data are acquired through either linear or rectangular arrays of detectors that must be periodically and systematically refreshed prior to the receipt of the next pixel of information. The SPOT satellite is the first commercial operational system of this type that uses this type of technology. Sensor fabrication technology trends indicate that there will be many new systems like the SPOT Image Corporation's satellite in the future.

Figure 3-3 illustrates the general trend of sensor packing and the number of pixels per scene to be expected in the 2000 A.D. time frame [Kostiuk and Clark 1984]. Tables 3-12 and 3-13 present some of NASA's projections in this arena for the same time frame. The number of pixels for the NASA projected Multispectral Linear Array are larger by two orders of magnitude when compared with Landsat TM. When the document was published in 1984 there was a lower limit on the pixel size by law. Since then there has been a change in the law. By presidential directive it is now possible for any remote sensing operation in the USA to acquire, process, and distribute data with pixels as small as 5 meters on a side. This increases the quantity of information by another factor of 36. (Here the TM pixel is given as a 30 meter square. It would take 36 of the 5 meter pixels to fill a single TM pixel). Steps to implement this type of technology are underway for Landsat 7.

When using arrays the first step of data processing becomes one of handling a gain and an offset for each detector of the array. This requires a lot of memory just for the purpose of the radiometric correction. Consider, for example, a set of linear arrays that replace the detector/mirror combination for Thematic Mapper. This would require that there be on the order of 6200 pixels per array per band. Thus there would be the requirement to carry $7*6200 = 43,400$ pairs of gain and bias data parameters. These 90,000 words would be required for a nominal calibration. Using Landsat gain and bias adjustment parameters for each detector for postlaunch upgrades would introduce the requireement for an additional 90,000 words as would any post launch recalibration.

New detector materials and the use of linear or rectangular arrays make the problem more difficult. For these arrays it has been determined that a better calibration can be achieved through use of a piecewise linear fit to the digital count versus radiance curve. Each piece of this fit introduces on the order of 90,000 words for calibration. Evidently this leads to a trade off between the memory size, the cycle time, and the radiometric accuracy calculations.
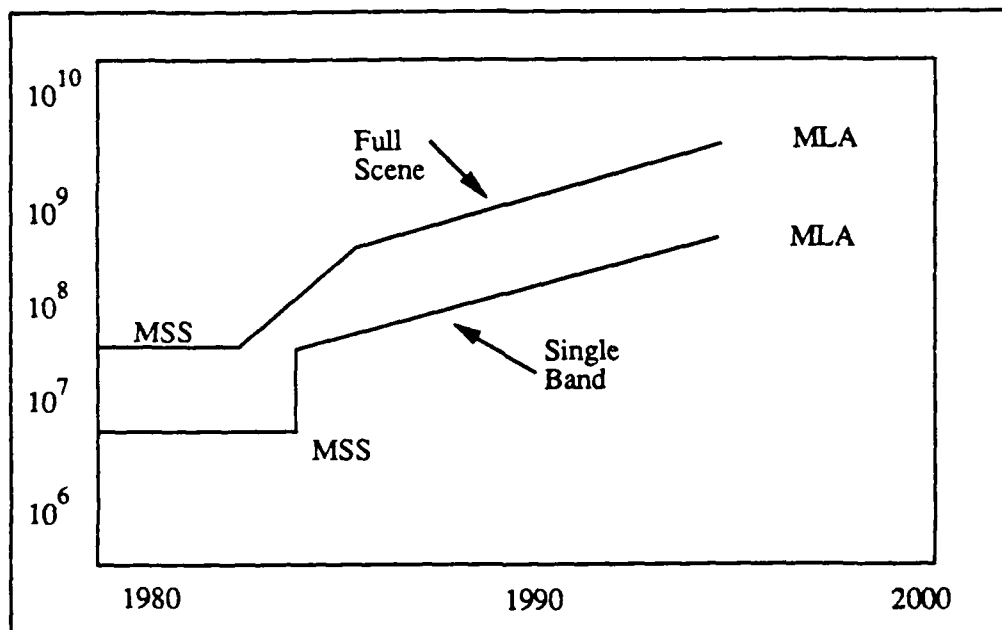
Figure 3-3. Projection of Single-Band and Full-Scene Data
Processing Requirements for the Future (LANDSAT)

Table 3-12. Multispectral Linear Array Potential Sensors

Pointable High Resolution Imaging Radiometer (PHRIR)
Supports experiements requiring the highest spatial resolution, off Nadid viewing, modest field-of-view, and high spectral resolution with bands defined prior to flight
Pointable Imaging Spectrometer (PIS)
Supports experiments requiring in-orbit selection from many high resolution spectral bands, moderate spatial resolution, small field-of-view
Moderate Field-of-View Imaging Radiometer (MFIR)
Supports experiments requiring 3-5 day coverage of large areas with modest spatial resolution and a limited number of spectral bands
Wide Field-of-View Imaging Radiometer (WFIR)
Supports experiments requiring 1-2 day coverage on large areas with low spatial and spectral resolution
Thematic Mapper (TM)
This sensor has flown on LANDSAT 4 and is shown for reference purposes

Table 3-13. Multispectral Linear Array Potential Sensors' Performance

	PHRIR	PIS	MFIR	WFIR	TM
Field-of-View (km)	60-180	10-30	200-800	1000-2000	185
Instantaneous F-o-V VIS/NIR SWIR MIR TIR	5-20 10-40 20-80 40-160	30-60 30-60 60-120 60-120	50-200 100-400 200-800 400-2000	300-600 300-600 300-600 300-600	30 30 -- 120
Spectral Resolution VIS/NIR SWIR MIR TIR	20 20 100 100	10-20 10-20 50-100 50-100	100 100 200 500	200 200 400 1000	80 200 -- 1200
Number of Bands Available VIS/NIR SWIR MIR TIR	8 8 4 4	30-6 100-200 8-16 40-80	3 2 2 2	3 2 2 2	4 -- 2 1
Number of Bands Transmitted	TBD	TBD	All	All	All
Data Rate (Mb/s)	300	TBD	<30	<10	85
Pointing Capability	Yes	Yes	No	No	No
Complexity	High	Very High	Moderate	Low	High

The second example involves the implementation of this type of system for the polar platform. Consider the High Resolution Imaging Spectrometer (HIRIS). This system uses 192 channels rather than the 7 bands for TM. The downlink data rate is expected to be on the order of 300 Mbytes/sec roughly 3.5 times faster than that used by TM. Use of one-half the capacity of this system on a constant basis is expected to generate data volumes that exceed those of TM by approximately a factor of three. Obviously the data rates and volumes are increasing quickly the area detectors planned for HIRIS will require over 5 megabytes of memory for their calibration. At this time there is no assured methodology for processing this data stream. Therefore, non-von Neumann architectures are under consideration.

3.6.2 Potential Futures Uses for NvN Architectures

One potential use of NvN architectures could be for the calibration of the 100 detectors ..Each detector could have its own processor set. Data could be pipelined into processors where a multiplier and adder for each detector could be applied to pixels as they crossed the appropriate time boundary. (This could be done on a per band basis or on a per line basis). Geometric manipulation requires significant use of matrices. If the data were properly rasterized the problems inherent in the slow throughput for geometric processing would be diminished.

Some NvN architectures are particularly suited to the extraction and processing of GCP. These include Fourier transform machines, neural network machines, and other pattern recognition schema that are compatible with NvN architectures. However, this area should be approached with some caution. At the most recent conference on Artificial Intelligence and Expert Systems hosted by the Central Intelligency Agency (October 1988) it was noted that when over 1000 nodes were used in the pattern recognition task the computer was given a capability roughly equivalent to that of a common housefly. This is the state of the art today. In order to consider pattern recognition in context of human vision it will be required that the number of nodes be more than 10,000,000. In addition, it was pointed out at the conference that NvN architectures such as neural networks would have excessive compute times as the number of connections increases. Some researchers stated that in order to do their processing they used Cray computers in unique configurations. The complexity of the task of pattern recognition becomes even more interesting when multispectral data are used.

3.6.3 Matching Architectures

None of the existent architectures have yet been proven able to ingest the raw data in a bit serial format and process to a final product. Most of the architectures given deal with the data after it has been processed to a medium (CCT) that is compatible with ingest and data manipulation. However, there are candidate architectures for the future that are under study for multiple applications of NvN systems. For example the hybrid system being installed by the U.S. in Alaska for the processing of SAR data is a partially NvN architecture. It is simpler than that required for Landsat since it does not deal with multiple spectral bands. Normally SAR systems deal with one or at most two frequencies and polarizations. Each could be pipelined separately as required but must be made compatible through preprocessing with the pertinent NvN architecture.

The Aliant system is under consideration by JPL for the processing of HIRIS data. It is their conjecture that this type of system can generate data at the required rate with the minimum of risk. Since their first PDR they have had to scale back somewhat from their original concept of using seven systems of this type. Nonetheless, they are progressing with this as one of their baseline system configurations.

Systems such as neural networks must be very carefully scrutinized. The human eye contains up to one million neurons [Cornsweet 1970]. These are currently thought to be cross-strapped to one another in localized bundles as required for pattern recognition tasks. Emulation of these neurons through neural networks is a new technology for hardware and software engineering. The Connection Machine and other configurations that use multiple emulated neurons can at present be used only for the simplest of pattern recognition tasks. There has been no way for these systems to function with over one million nodes and one million factorial connections within the existent state of the art.

Table 3-14 evaluates NvN systems for image processing.

Table 3-14. Evaluation of NvN Systems for Image Processing

MACHINE	ENHANCEMENT	RESTORATION	ANALYSIS
Alliant	Yes	MIMD	Yes
Anza-Plus	Marginal	?	Marginal
ASPRO	No	SIMD	No
Boltzman Machine	No	?	Marginal
BSP	Marginal	SIMD	Yes
Butterfly	Yes	MIMD	Yes
CSC Star 100	No	SIMD	No
CEDAR	No	MIMD	No
Celerity	No	SIMD	No
CHiP	No	MIMD	No
Cm* Machine	No	MIMD	No
CMOS VLSI	No	?	No
Connection Machine	Yes	SIMD	Yes
Convex C-1	Yes	MIMD	Yes
Cosmic Cube	Marginal	MIMD	Marginal
Cray	Marginal	MIMD	Marginal
Cray-1	Marginal	SIMD	Marginal
Cyber 205	Marginal	SIMD	Marginal
Cyberplus	Yes	MIMD	Yes
DADO2	Yes	SIMD/MIMD	Yes
DAP	Yes	SIMD	Yes
Data Driven Machine	No	?	Yes
ELI	No	?	No
ELIXSI	?	MIMD	?
Encore Multimax	Marginal	MIMD	Marginal
ETA-10	Yes	MIMD/(M)SIMD	Yes
FACOM	No	SIMD	No
FGCS	?	?	?
FLEX/32	No	MIMD	No
Galaxy	No	SIMD	No
HEP	No	MIMD	No
Hitachi	Marginal	SIMD	Marginal
Illiac IV	No	SIMD	No
iPSC	Marginal	MIMD	Marginal
Matrix-1	Yes	SIMD	Yes
MIT Dataflow	No	MIMD	No
MPP	Yes	SIMD	Yes
NEX SX-2	Marginal	SIMD	Marginal
NETL	No	NA	No
Neural Phonetic	No	NA	No
NON-VON	Yes	MIMD/(M)SIMD	Yes
PASM	No	SIMD/MIMD	Yes
STARAN	Yes	SIMD	Yes
Systolic Adaptive	No	MIMD	No
Systolic Cellular	Marginal	SIMD/MIMD	Marginal
T-ASP	Yes	SIMD	Yes
Tagged Token	No	MIMD	No
TI-ASC	Marginal	SIMD	Marginal
TRAC	Yes	SIMD/MIMD	Yes
Ultracomputer	No	MIMD	No
WARP	Yes	MIMD	Yes

3.7 GENERAL PURPOSE USE OF NVN MACHINES

3.7.1 Use in Development, Prototyping, and Testing of Hardware and Software

This section discusses the use of NvN architectures in accomplishing the following:

- Developing Production-Grade Software

- Prototyping Software Systems

- Testing of Hardware and Software.

The technical literature indicates that the overwhelming majority of reported usages of NvN machines in developing software have come from the scientific and engineering computation arena; FORTRAN is the most commonly used language for such computations, although there is increasing usage of C and Pascal. The literature does not indicate the use of NvN machines for developing software that is applicable to the BM/C^3I domain.

There are no reported instances of using NvN machines for the prototyping of software systems. However, the AI community makes heavy use of the prototyping paradigm, and it is therefore possible to consider the development of AI production systems on various NvN architectures as examples of prototyping. There are no known examples of using NvN machines as a hardware base for system(s) that support/facilitate the prototyping of real-time systems or other complex applications that are typical of the BM/C^3I domain.

There are no literature references to usages of NvN machines in the testing of hardware and software.

Although there has been little or no use of NvN architectures for developing software of production-grade other than in the scientific and engineering communities, or for prototyping software systems, or for the testing of hardware and software, some of the NvN architectures, but particularly the Class VI machines, would be excellent foundations for a comprehensive system and software engineering environment, such as that needed to support the development of BM/C^3I applications.

A comprehensive system and software engineering environment (S/SEE) that is suitable for developing BM/C^3I applications must be supported by a database which will grow from a moderately large size to an extremely large size over the course of a software development cycle, during which the S/SEE will have to support all three task types mentioned in the first paragraph above.

The S/SEE's very large database would be distributed over multiple storage devices, and the responsibility for managing particular devices, or sets of devices, could be assigned to different processors within an NvN machine instance. Managing the S/SEE database would, very probably, be greatly simplified if the data were distributed across the sets of storage devices according to the

various views about the data that are characteristic of the different job-related roles in a software development community. For example, the requirements engineer, the system design engineer, the software design engineer, the programmers, the test engineer, and the project management staff all have rather different views about the contents of the database. A Class VI NvN machine would be, perhaps, a nearly ideal foundation for the S/SEE.

3.7.2 Problem Domains to Which NvN Architectures are Applicable

There are no problem domains to which all seven classes of NvN architectures are "best" applicable. Their very different natures makes their applicabilities very different. To a great extent, the discussions in Section II of this report have dealt with the issue of domain applicability; therefore, this section will present only a top-level summary.

Class I NvN machines have the broadest applicability because they are very similar to TvN machines, which have been used for every conceivable kind of computational or data processing job. The vector capabilities of this architecture are very commonly applied to tasks such as seismic modeling, and fluid dynamics.

Class II NvN machines are best applied to algorithms that perform regular, predictable calculations, such as matrix operations involved in signal processing. In machines of the wavefront subclass, the global synchronizing clock (characteristic of systolic machines) is replaced by dataflow principles. Reported applications include radar and sonar signal processing, nucleic acid sequence comparisons, and linear algebra.

Class III machines are generally applied to the same kinds of scientific and engineering applications to which Class I machines are applied. The bit-plane-oriented subclass.order are particularly suitable for signal processing; other reported applications include:

- satellite imagery and data processing
- numerical analysis
- Monte Carlo simulations
- solving partial differential equations
- nuclear energy modeling
- seismic data processing
- structural analysis
- economic simulations.

Class IV machines are most appropriate for applications involving contents-based searches of large databases. These machines have been used for tracking and surveillance, cartography, image processing, and signal processing.

Class V machines are still used primarily in research laboratories. The dataflow principles—enabling instructions when all operands needed for the instruction have been made available—have been applied to systolic architectures to create a variant architecture, called wavefront machines.

The set of Class VI machines covers a broad spectrum of architectural features. It is highly likely that Class VI machines will be used for virtually every kind of application. Within the BM/C³I domain, the first applications to which these machines will be applied are tracking and surveillance and large database management. Class VI, together with Class I, machines will likely become the workhorses of the BM/C³I arena.

Reported applications for Hypercube Topology with PE-to-PE communications include: astrophysics, quantum chemistry, fluid dynamics, and structural mechanics.

Reported applications for Ring Topology with PE-to-PE communications include: digital signal processing, and scientific and engineering computation and data processing.

Reported applications for Tree Topology with PE-to-PE communications include:

- AI production systems
- database applications
- mathematical and scientific applications
- image and signal processing
- speech recognition.

Reported applications for Reconfigurable Topology with PE-to-PE communications include:

- simulations
- Fast Fourier Transform-based computations
- image processing
- various AI applications.

Reported applications for Bus Interconnection, PE-to-Memory communications include:

- simulations
- seismic data processing
- various aerospace applications
- image and signal processing.

Reported applications for Direct Memory Access Interconnection, PE-to-Memory communications include:

- a wide variety of scientific and engineering applications

- simulation and modeling
- numerical analysis.

The applications that have been reported for Multistage Interconnection Network, PE-to-Memory communications architectures indicate that these machines are intended to be, and are being used as, general purpose machines.

Class VII machines are found primarily in academic and industrial research and development laboratories. The concepts of neural nets are still so new and strange that it is not possible to make any accurate predictions about domains to which they might be applied. The reported applications include:

- speech recognition
- associative memory processing
- alphanumeric character recognition.

CHAPTER IV. SOFTWARE ENGINEERING FOR NON-VON NEUMANN ARCHITEC-
TURES

4.1 SOFTWARE ENGINEERING ASSESSMENT

To provide a framework for discussion of the current state-of-the-art of software tools and techniques within the context of Non-von Neumann architectures, one must first consider the concept of a "programming environment".

A programming environment is a set of tools used to support program development. Although this description is (of necessity) vague, it is not ambiguous. In particular, given an architecture, a language, and a type of problem, the desirability of a tool with some specified functionality can be determined. For example, if the programming language were interactive (e.g., PAR-LISP) an interpreter would be desirable, whereas the need for an interpreter would be questionable for a language such as parallel FORTRAN. Any tool, of course, must provide some functionality. To further structure this analysis, any particular tool will be analyzed within the context of the software development life cycle. An ideal programming environment would provide a user with a homogeneous interface to a set of tools aiding in program development through the entire software life cycle. In this manner, it will be possible not only to categorize available tools and tools under development, but will also be possible to define tools that are needed because of the role they could play in helping to automate software development.

Although much work is being done on models of software development, the following high-level components are common to all software development life cycles: requirements analysis, design, coding and unit testing, integration and test, deployment, and maintenance. These components ("phases") may be related to each other in different ways (as in, for example, the "waterfall" or "spiral" model), but they remain constant constituents of any life cycle. Indeed, the relationships defined and/or permitted between phases determines the particular life cycle model or methodology.

The purpose of each phase is as follows:

Requirements analysis—This includes a systematic review of the real and perceived needs for information and the definition of the system to be analyzed; a detailed study of the administrative and operational systems in an organization, and the specification and gathering of the information necessary to fully comprehend and solve the problem under consideration; the organization and documentation of such information; the identification of alternative methods of approaching the solution and the feasibility thereof; and the logical specification of the major functions and subfunctions of the software system and their relationships.

Design—This is a phase intermediate between requirements analysis and coding and unit testing. The results of the requirements analysis phase (which is at a high logical level) are here brought down

to a more detailed level, providing a description detailed enough to allow programmers to implement the proposed solution. It should be noted that the boundaries between the requirements analysis phase and the design phase are fuzzy, with some logical specifications generated during the analysis phase being sufficiently precise to allow implementation. In general, however, for large, complex software systems, the analysis phase is at too high a logical level to permit implementation. In general, one may say that the requirements analysis phase results in a logical specification ("what" is to be done) while the design phase results in algorithmic specifications ("how" things should be done).

Coding and unit testing—The implementation of code for software systems components, including a thorough debugging of the components. This phase implements the algorithms of the design phase into the designated programming language, under the constraints imposed by the particular language and the hardware upon which the program is to be executed.

Integration and test—The integration of the various components as implemented in the prior phase, and extensive testing of the total system.

Deployment—The deployment of a developed and tested complete software system into the field for everyday use.

Maintenance—The maintenance of a deployed system, including bug fixes, enhancements, and user support.

Various documentation may be required for each phase, both within the phase and between phases. Strictly speaking, however, such documentation is not part of the development life cycle but is an addition to it to ensure proper and complete understanding of the various phases and of the software system as a whole. Note that DoD-STD-2167 and 2167A do require such documentation. The requirement of documentation is to satisfy a need for communication, both among participants within a phase and between participants of different phases. Such communication, is not, part of a particular methodology. This is analogous to the situation with respect to "traceability". Traceability is not a requirement of a high-level model of a life cycle, but of the instantiation of a particular methodology. Both documentation and traceability tools are very desirable, but this difference between a high-level model of a life cycle and a particular methodology must be kept in mind.

Different models of the life cycle incorporate the above phases differently. For example, DoD-STD-2167 includes both preliminary design and detailed design as separate components of the software development life cycle. The above phases are life cycle components at an appropriate level of detail for the study undertaken for this report. In particular, they can be refined and/or partitioned into components (as in 2167) or taken as the highest-level partitioning of any arbitrary life cycle, i.e., any life cycle methodology must include these phases. By considering the software development life cycle from this vantage point, the potential automation of the life cycle for non-von Neumann machines can be analyzed independently of any particular life cycle methodology. Such a vantage

point excludes the interrelationships of the phases: each phase must be executed in full, without knowing the details of the interrelationships of the phases, although a phase may be partially completed before another phase is begun, that latter phase providing feedback to a prior phase. Such, of course, would occur in the course of rapid prototyping or the development of AI-based software systems. Such a high-level view also is amenable to such alternative models of software development methodologies as the spiral model by the appropriate definition of these inter-phase relationships.

Within each phase, those tools which would support the automation of that phase may be defined, under the assumption that inter-phase communications are not under consideration for automation at this stage. The particular tools necessary to completely automate the life cycle, including inter-phase linkages, would be a function of the particular software life cycle methodology to be implemented. Again, by separating the phases as above, this issue can be deferred with no effect on the analysis to be performed in this report.

4.1.1 Life Cycle and NvN Architectures

Examination of the software life cycle phases, independent of a specific methodology, indicates they are applicable to any software development effort. At this level of abstraction, the software life cycle phases are also applicable to software development for NvN architectures. Introducing NvN machines into software development efforts may cause the scope of a phase's task to change; but, the purpose of each life-cycle phase should remain unaffected.

Class VII, Neural Network, machines represent a potential anomaly to the software development life cycle model. The limited amount of data on "real" neural network machines precludes any final conclusions. In theory, neural network machines are trained, not programmed, to perform a specific task; hence, there is no software that needs to be developed for a neural network to perform its task. However, neural networks will probably be a component of a larger system and perform a specific function; so, software will need to be developed to support the inclusion of a neural system. In the future, development of neural network systems will probably consist of selecting training sets and choosing a learning methodology. Therefore, the following comments may not apply to machines in Class VII.

In general, the greatest effect NvN architectures introduce into a system is increased complexity. For this reason, NvN systems should be selected based on performance requirements. Traditional von Neumann computers employ essentially the same architecture independent of manufacturer. Design and implementation decisions can greatly impact performance; however, introducing the same effect on any von Neumann machine will result in similar performance characteristics. For NvN machines this is not necessarily the case, implementing the same algorithm on different architectures can have a significant effect on performance. Thus, an evaluation of the life cycle phases assumes that the activities will be identical to those for von Neumann machines, and discusses potential areas of increased complexity.

Requirements analysis—The purpose of requirements analysis is to completely define and analyze the requirements of a system. The generic task of understanding a problem and specifying what needs to be accomplished is applicable to any software development effort. A common element of requirements analysis is to perform feasibility studies, and make recommendations. A common technique used to perform this task is prototyping potential solutions. NvN architectures could add complexity to prototyping and feasibility analysis. Potential problems with simulating processor-to-processor communications or memory bottlenecks caused by multiple processor systems will make prototyping more difficult.

Typical tools that one might desire for performing requirements analysis are problem definition languages, interactive/expert feasibility analysis tools, data and process modeling tools, prototyping tools, consistency checkers and completeness checkers. Such tools would all be useful in requirements analysis for NvN systems. Modeling and prototyping tools would need to account for NvN architectures.

Design—The purpose of the design phase is to produce a complete detailed design, or blueprint, for all software development. NvN architectures could impact activities associated with this phase of the life-cycle. A detailed design includes selection of algorithm(s) and implementation language(s), two key aspects in determining system performance. The potential problem of determining an algorithm's performance on a previously untested architecture is difficult. Improper algorithm selection could reduce performance significantly.

Useful tools for performing design are data flow analysis, PDL processors, alternative design impact, performance analysis estimators, and simulators. Such tools would be useful for NvN architectures and would have to account for NvN machines in functionality.

Coding and unit test—The purpose of the implementation phase is to code and unit test all the software modules specified in the detailed design document. In addition to the increased complexity introduced by NvN architectures, the definition of a code unit must be expanded. A common technique used to implement algorithms on NvN machines involves replicating a code segment and distributing data among available processors. This technique often requires these replicated code segments to communicate, and hence, unit testing is more complex. The complexity is increased by introducing potential timing errors and a non-deterministic order of events.

Commonly used tools for coding and testing are languages, compilers, linkers, editors, optimizers, test generators, and debuggers. Such tools are useful for NvN machines; however, each tool would probably have to be developed for a specific architecture or machine.

Integration and test—The purpose of testing is to verify that a completed system satisfies the specified requirements. There should be no effects from NvN architectures on this phase of the life-cycle, other than adding complexity by including multiple processors.

Typical tools that one might expect for integration and test are test case generators, diagnostic tools, performance analyzers, static analyzers and output comparators. Such tools are needed for NvN systems.

Deployment—Deployment of software systems for NvN architecture should be essentially identical to deployment for standard von Neumann systems. One potential problem could arise if the development hardware is not identical in number of processors and PE-to-PE communications bandwidth to that of the deployed hardware. Differences in these components, or other components, may have detrimental effects on system performance.

Maintenance—The purpose of the maintenance phase is to correct any detected errors or inconsistencies and enhance an existing systems functionality, as required . NvN architectures introduce a degree of complexity to this phase. Side-effects of altering a single code-fragment could alter timing or introduce inconsistencies in remote processors. Again, this is due to the potential complexity of software on NvN machines, which can have concurrently executing processes on potentially remote processors.

Useful tools for maintenance are similar to the tools needed for coding and unit test; languages, compilers linkers, editors, optimizers, test generators, debuggers, diagnostic tools, performance analyzers, static analyzers and output comparators. Such tools are useful for NvN machines; however, each tool would probably have to be developed for a specific architecture or machine.

As the survey information in section 4.2 shows several tools are under development which automate some aspect(s) of a life cycle phase. A proliferation of tools are for the coding and unit test phase of the life cycle. Tools such as compilers, operating systems, languages and debuggers are the most common tools under development.

As stated previously, the purpose of each phase of the life cycle is expected to remain unchanged, while specific tasks in a phase will be extended to account for NvN architectures. The automation of such tasks in each phase will require the development of tools that account for NvN systems. As methodologies evolve and inter-phase dependencies are defined, the impact of NvN systems might be more apparent.

Two research areas that include several tools under development are programming models (e.g., LINDA and Paralation) and graphics oriented programming tools (e.g., SCHEDULE, CODE and CODE UCG). Such tools attempt to integrate the design and implementation phases of the life cycle. The close association between architecture and algorithm on performance might require a tight interaction between the design and implementation phases. A possible trend for NvN machines might be a tight coupling of these two phases for the development of efficient, high performance algorithms.

4.2 SOFTWARE ENGINEERING TECHNOLOGY ISSUES

This section discusses software engineering technology issues that are applicable to NvN architectures. Information reviewed was obtained through literature surveys, discussions with vendors, and discussions with users, particularly users in academia. The current focus in NvN software technology is on tools that automate some aspect of the application development process. Section 4.2.1 presents the information gathered during the survey of software engineering tools for NvN architectures. Section 4.2.2 presents an analysis of the collected data. Existing software tools are examined for their applicability to NvN architectures and to the development of application programs. The survey information includes the current status of state-of-the-art software tools for NvN architectures.

4.2.1 Software Engineering Tools for NvN Architectures

The survey unveiled work in the following areas: operating systems, code optimization tools, programming languages, debuggers, performance monitors, programming models, and hardware simulators. These areas of work comprise the majority of research in NvN computing software tools, and therefore, are the subsections in this report. Many of these tools could be incorporated into a programming environment. A few research institutions have put together tool sets that could be the core of a programming environment.

4.2.1.1 Operating Systems for NvN Architectures

An operating system provides a foundation upon which a software engineering environment can be developed. The technical literature clearly indicates that UNIX is becoming the operating system of choice for NvN machine users. Table 4-1 identifies operating systems used on various NvN machines of various architectures. Thirteen of the twenty-five operating systems presented are based on UNIX.

Cosmic Environment and Reactive Kernel—The Reactive Kernel is a portable multicomputer operating system developed at the California Institute of Technology. The Cosmic Environment is a portable host run-time system for use with the Reactive Kernel. The Cosmic Environment provides a set of processes, utility programs, and libraries to support communications between host and node processes. The Reactive Kernel node operating system supports multiprogramming, message-driven process scheduling, storage management, and system calls for message passing. Together they provide uniform communication between processes independent of the node on which they are located. Currently, the Cosmic Environment and Reactive Kernel are available on Symult and Intel systems.

MACH—MACH was developed at Carnegie Mellon University to support parallel processing. MACH is an operating system designed for parallel architectures. MACH supports asynchronous processes through multiple threads of control. Semaphores are supported to provide concurrent processes a synchronization mechanism to pass or share information.

Table 4-1. Matching Machine Operating Systems to Literature Citations

OPERATING SYSTEM	MACHINE NAME	LITERATURE CITATION
Armstrong O/S	Armstrong Multicomputer	[Rayfield, *et al*. 1988]
Chrysalis	BBN Butterfly	[Miller 1988]
MACH	BBN Butterfly	[Thomas 1988]
Concentrix	Alliant FX/8	[Miller 1988]
COS, UNICOS	Cray X-MP/4	[Miller 1988]
DYNIX	Sequent	[Sequent Computer]
EMBOS	ELXSI System 6400	[Miller 1988]
MMOS	FLEX/32 Multicomputer	[Miller 1988]
PASMOS	PASM	[Siegel 1987]
PEACE	Supernum	[Schroder 1988]
Psyche	BBN Butterfly	[Scott 1988]
Sprite	SPUR (UCB)	[Ousterhout 1988]
Reactive Kernel	Symult 2010	[Athas and Seitz 1988]
TOS	T-ASP	[Lang, *et al*. 1988]
Trollius	Transputer	[Cornell University]
Uniform	BBN Butterfly	[Thomas 1988]
UMAX	Encore Multimax	[Miller 1988]
UNIX	ASPRO	[Miller 1988]
UNIX	Convex	[Convex Computer]
UNIX	ELXSI System 6400	[Dongarra, *et al*. 1987]
UNIX	ETA-10	[Dongarra, *et al*. 1987]
UNIX System V	FLEX/32	[Dongarra, *et al*. 1987]
Xylem	CEDAR	[Miller 1988]

MMOS—MMOS was developed by Flexible Computer Corporation to support real-time applications on its shared memory NvN architecture computer. MMOS provides semaphores for synchronizing concurrent processes through operating system calls. These operating system calls are supported directly by the Flex/32 hardware.

PEACE—Process Execution And Communication Environment (PEACE) was designed and built for the Supernum supercomputer. PEACE is a process-oriented operating system for message passing processor-to-processor communication architectures.

Psyche—Psyche attempts to provide a high-performance operating system to support a wide variety of non-uniform memory access (NUMA) machines. The research is conducted on a BBN Butterfly computer. The Psyche model assumes that shared memory and message-passing are relatively equal with respect to usability, with the application dictating which is more appropriate. Four basic concepts comprise the Psyche model:

1. realm—data and access protocol
2. thread—control flow and scheduling
3. protection domain—access relationships
4. keys and access list—controls access to processes.

Trolius—Trolius was developed at Cornell University and is intended for MIMD architectures. Trolius is a high level operating system that attempts to provide a consistent development environment for application development. Trolius provides an interface between a computer's operating system and application programs through I/O and communications systems calls.

4.2.1.2 Code Optimization for NvN Architectures

Developing applications programs for NvN machines in a particular high order language (HOL) requires compilers that can exploit the architectural feature(s) that characterize an NvN architecture class. Table 4-2 identifies conventional HOL compilers that either restructure sequential code into parallelized sequences or that handle compiler directives that provide for parallel operations. Table 4-3 identifies code restructuring tools that aid in transforming FORTRAN 77 to a parallel form of FORTRAN, or some intermediate representation of a program acceptable to a compiler (e.g., compiler directives inserted for a FORTRAN compiler).

Table 4-2. Available Optimizing Compilers

MACHINE NAME	LANGUAGE	LITERATURE CITATION
Alliant FX/8	C	[Dongarra 1987]
Alliant FX/8	FORTRAN	[Argonne 1988]
Alliant FX/8	PASCAL	[Dongarra 1987]
Amdahl VP-E Series	FORTRAN	[Argonne 1988]
Ardent Titan-1	FORTRAN	[Argonne 1988]
ASPRO	FORTRAN	[Miller 1988]
ASPRO	OPS-83	[Lott 1987]
BBN Butterfly	C, LISP, FORTRAN	[Miller 1988]
CDC CYBERPLUS	FORTRAN	[Dongarra 1987]
CDC Cyber 205	FORTRAN	[Argonne 1988]
CDC Cyber 990E/995E	FORTRAN	[Argonne 1988]
Cedar	FORTRAN	[Miller 1988]
Celerity 6000	FORTRAN 77	[Miller 1988]
Connection Machine	C, LISP	[Hillis 1985]
Convex C Series	C	[Miller 1988]
Convex C Series	FORTRAN	[Argonne 1988]
Cray Series	FORTRAN	[Argonne 1988]
Cray Series	FORTRAN	[Argonne 1988]
Cray X-MP	C	[Miller 1988]
Cray X-MP	FORTRAN	[Argonne 1988]
Cray X-MP	FORTRAN	[Myers 1986]
Cray X-MP	PASCAL	[Miller 1988]
Cray-2	FORTRAN	[Argonne 1988]
DADO2	C, LISP	[Stolfo 1987]
DAP	FORTRAN	[Miller 1988]
Encore Multimax	C, FORTRAN 77	[Miller 1988]
Encore Multimax	PASCAL	[Dongarra 1987]
ETA-10	FORTRAN	[Argonne 1988]
FACOM VP-200	FORTRAN 77	[Miller 1988]
FLEX/32	C, FORTRAN, RATFOR	[Dongarra 1987]
Gould NP1	FORTRAN	[Argonne 1988]
Hitachi S-810/820	FORTRAN	[Argonne 1988]
IBM 3090/VF	FORTRAN	[Argonne 1988]
IBM 3090/VF	FORTRAN	[Liu 1988]
Intel iPSC	C, LISP	[Miller 1988]
Intel iPSC/2-VX	FORTRAN	[Argonne 1988]
Intel iPSC	PROLOG	[Dongarra 1987]
MPP	PASCAL	[Potter 1985]
NEC SX/2	FORTRAN	[Argonne 1988]
Saxpy Matrix-1	FORTRAN	[Foulser 1987]
SCS-40	FORTRAN	[Argonne 1988]
Stellar GS 1000	FORTRAN	[Argonne 1988]
Ultracomputer	C, FORTRAN, PASCAL	[Gottlieb 1983]
Unisys ISP	FORTRAN	[Argonne 1988]

Table 4-3. Matching Restructuring Tools to Literature Citations

Tool	Citation
ART	[Applebe, et.al. 1985]
KAP	[Padua, et.al. 1986]
Parafrase II	[CSRD 1987]
PAT	[Smith, et.al. 1988]
PFC	[Allen, et.al. 1982]
PTOOL	[Carle, et.al. 1987]
Sigma	[Guarna, et.al. 1988]

ART—ART (Anomaly Reporting Tool) is a static source code analyzer for parallel FORTRAN source code [Appelbe and McDowell 1985] that constructs a "concurrency history" of a program in order to detect the following classes of potential bugs (anomalies): (1) references to variables which have values depending on task scheduling; (2) deadlock and busy-waiting loops; (3) race conditions. The tool is geared to vector architectures and numeric applications, although it can be generalized to support shared memory vector architectures by parameterizing synchronization primitives.

KAP—KAP is an automatic vectorizing precompiler that transforms FORTRAN 77 code into FORTRAN 8X. KAP is built by Kuck and Associates, Inc.; it is available for several different computers including Cyber 205, Alliant, NEC. KAP is capable of restructuring code for vector operations.

Parafrase II—Parafrase II, a source code parallelizing tool, was developed at the University of Illinois Center for Supercomputing Research and Development (CSRD). The tool transforms FORTRAN or C source code to produce new HOL code that is more easily parallelized.

PAT—PAT (Parallelizing Assistant Tool), being developed at the Georgia Institute of Technology, is a more advanced version of the ART source code tool [Smith and Appelbe 1988]. This interactive tool determines and displays loop dependencies. At the user's direction, PAT can perform source code transformations that increase parallel execution efficiency, employing a large library of such transformations.

PFC—The PFC (Parallel FORTRAN Converter) tool, developed at Rice University [Allen and Kennedy 1982], transforms FORTRAN source code to create vectorizable code. Sophisticated analysis of data dependencies is performed, emphasizing potential vectorization of subscripted array references in control loops.

PTOOL—PTOOL, an interactive source code analyzer, was developed at Rice Universi y [Kennedy, et al., 1987]. PTOOL uses a database of inter-statement dependencies created by PFC analysis. The tool analyzes loops in sequential FORTRAN programs; if dependencies prevent the parallelization of a loop, the tool displays to the user the problematic dependencies and then explains why they prevent parallelization.

SIGMA—SIGMA, an interactive tool, was developed at the University of Illinois CSRD for parallelizing FORTRAN, BLAZE and C programs [Guarna, Gannon, Gaur and Jablonowski 1988]. SIGMA identifies data dependencies and aids in identifying legal code transformations to exploit parallelism and in specifying parallel subroutine execution. It is intended to accommodate large programs and is linked to parallel application performance tools developed at the CSRD. This tool was originally termed "BLED", for BLAZE Editor [Gannon, Atapattu, Lee and Shei 1988].

4.2.1.3 Programming Languages for NvN Architectures

With the advent of NvN machines came opinions on how best to make use of them. A majority of users regards the development of new HOLs as essential to utilizing these machines. Table 4-4 identifies twenty-nine languages that are used for parallel programming.

Table 4-4. Matching HOLs and Literature Citations

LANGUAGE NAME	LITERATURE CITATION
Actus II	[Perrott, et al. 1987]
AL	[Webb 1989]
APPLY	[Webb 1989]
Blaze	[Keolbel, et al. 1987]
Cantor	[Athas, et al. 1988]
Concurrent Pascal	[Hansen 1975]
CONSUL	[Baldwin 1987]
CSP	[Segall, et al. 1985]
C*	[Hillis 1985]
Data Flow Language	[Gokhale 198x]
DURRA	[Barbacci 1987]
Edison	[Segall, et al. 1985]
FORTRAN PLUS	[AMT]
Glypnir	[Welch 1984]
Id	[Arvind, et al. 1978]
*Lisp	[Hillis 1985]
MAINSAIL	[Dongarra 1987]
MDFL	[Kung, et al. 1982]
Mesa/Cedar	[Swinehart, et al. 1985]
MLP	[Lang, et al. 1988]
MP	[Segall, et al.1985]
Multilisp	[Halstead 1986]
Multi-Pascal	[Lester, et al. 1987]
Occam	[Wayman 1986]
Parallel Pascal	[Reeves 1984]
Parallel PL/M	[Stolfo 1987]
Parallel PSL Lisp	[Stolfo 1987]
ParMod	[Eachholz 1987]
PARPC	[Martin 1987]
PISCES FORTRAN	[Pratt 1987]
PROTRAN	[Rice 1983]
VAL	[Dennis 1984]
Vector C	[Li, et al. 1985]
VECTRAN	[Paul 1984]
W2	[Gross, et al. 1986]
XX	[Snyder 1984]

Actus II—Actus II is a Pascal-based language for processor array architectures [Perrott 1987]. The language supports grid array data structures and is independent of the number of PEs. Actus II contains constructs for accessing elements of parallel arrays.

AL—AL, a programming language for systolic arrays, was developed at Carnegie-Mellon University [Webb 1989]. User's AL source code declarations specify how the compiler should distribute arrays and computations across PEs; arrays can be distributed by column or by row. AL is aimed primarily to support scientific and, particularly, signal processing applications.

APPLY—APPLY is a programming language for image processing applications [Webb 1989]; it is primarily geared to systolic array and array processor architectures. Users specify functions that are to be applied to a pixel and its surrounding region. The APPLY compiler distributes such regional functions across PEs. Currently, compilers exist for the WARP, IWARP, Computing Surface, and SLAPP architectures.

BLAZE—BLAZE, a Pascal-based language, is targeted for programming parallel applications on both SIMD and MIMD architectures [Mehrotra 1987]. The BLAZE "FORALL" statement provides a mechanism for achieving parallel execution.

CONSUL—CONSUL is a prototype constraint language. It is similar in appearance to Prolog or LISP. Its main data structure is the set, and it has set manipulation operators such as AND, OR, EXISTS, and FORALL.

C*—C* is a programming language available for the Connection Machine. It supports distributed data structures and operations on those structures.

Concurrent Pascal—Concurrent Pascal provides language extensions to handle the notions of "process" and "monitor" [Hansen 1975]. Emphasis is placed on synchronizing shared data access.

Concurrent Prolog—Concurrent Prolog extends Prolog by introducing or-parallelism and annotated read-only variables. Synchronization is supported when unification attempts to bind a read-only variable.

DURRA—DURRA is concerned with heterogeneous systems used in process control. It is a description language, where description refers to process communication and organization, not compilable, executable code. DURRA supports a message-passing model and provides constructs for starting tasks and reading and sending messages.

FORTRAN Plus—FORTRAN Plus extends FORTRAN 77 with array operators, as well as intrinsic functions for moving data. FORTRAN Plus introduces masks as array indices to provide a mechanism for accessing selected portions of an array.

Glypnir—Glypnir, an Algol-based parallel programming language, is targeted to the Illiac IV processor array [Welch 1984]. The language reflects the Illiac IV's mesh-structured Interconnection Network. Vector data types are explicitly declared for PEs. In addition, inter-PE operand routing is explicitly specified in the language.

ID—ID is a functional programming language developed by Arvind and his colleagues at MIT that is, in their words, "a declarative, implicitly parallel language that simultaneously raises the level of programming and reveals much more parallelism than is possible with programmer annotations".

Like other functional languages, ID exposes the parallelism inherent in the functional form. There are two aspects of ID that are particularly appealing: array initialization and loops.

Loops in ID are functional in the sense that there is not, conceptually, a set of variables that take initial values, then take next values, and so on. Instead, the interpretation of a loop is that there are, effectively, n (for appropriate n) copies of the loop body, each with one of the possible values associated with the loop variables.

*LISP—*LISP was developed for the Connection Machine. It introduces distributed data through xectors. A xector corresponds to a set of processors with a value stored in each processor.

MDFL—MDFL (Matrix Data Flow Language) uses a data flow graph notation scheme for programming wavefront array processors [Kung, S.Y. 1987].

MLP—MLP is a signal processing language for the Motorola T-ASP processor array [Lang, *et al.* 1988].

MP—MP is a meta-language defined for the Carnegie-Mellon University PIE environment [Segall 1985]. MP is intended for use with shared memory architectures. MP constructs include:

- activities—code collections (smallest schedulable unit)
- frames—declarations of shared data and operations, as well as monitoring operations on shared data access
- teams—contain specifications for parallel programming, and include activities, frames, and control information
- sensors—specify application monitoring code.

Multilisp—Multilisp versions based on LISP and Scheme have been produced for the BBN Butterfly and MIT "Concert" architectures [Halstead 1986]. This language uses "futures" constructs to serve as place holders for either values or data structures while a task calculates or constructs these elements. Explicit task delays can be specified by the user.

Multi-Pascal—Multi-Pascal is a Pascal-based programming language geared to MIMD architectures [Lester 1987]. It contains constructs for explicit parallel process creation and a parallel

execution FORALL statement. Multi-Pascal supports a message-passing model through the use of channel variables.

Occam—Occam is a language for fine-grained parallel programming of the Inmos Transputer; it is based on Hoare's CSP [Hull 87,Wayman 86]. It provides synchronized communications between transputer processes via "channels". Control flow constructs include sequential, parallel and "first component ready".

Parallel Pascal—Parallel Pascal is a parallel programming language for the Loral Massively Parallel Processors and similar SIMD architectures [Reeves 1984]. The language supports parallel array operations. A parallel WHERE statement is included to support selective data operations.

Parallel PL/M—Parallel PL/M is a system-level programming language for the DADO family of tree-structured architectures [Stolfo 1987].

Parallel PSL LISP—Parallel PSL LISP is a LISP dialect targeted to the DADO family of parallel architectures [Stolfo 1987].

ParMod—ParMod is a Pascal-based language geared to MIMD architectures [Eichholz 1987]. It supports parallel execution of modules that communicate through global procedures, and parallel task execution within modules.

ParLog86—ParLog86 extends Prolog by adding a parallel conjunction (and-parallelism) and simultaneous clause evaluation (or-parallelism). To provide implementors with a synchronization scheme, ParLog86 separates the two functions of unification, input and output.

PARPC—PARPC was designed for shared memory systems. The main parallel extension is "parproc" for invoking parallel procedures. The calling procedure is blocked while the parallel procedures are executing. Returned values are processed immediately and the procedure is blocked again until all procedure invocations respond. Communications are facilitated by IN and OUT parameters. PARPC programs have a single logical thread of control, but may execute many physical threads of control.

PFP—PFP is a FORTRAN-based language that supports parallelism at the loop, task, and subroutine levels [Forefronts 1988].

VAL—VAL is a single-assignment language for data flow architectures designed by Ackerman; VAL influenced Dennis' static M.I.T. Data Flow Computer [Treleaven 1982].

Vector C—Vector C is a parallel language for the Cyber 205 architecture. Its language extensions include vector data types, expressions, and operators.

VECTRAN—VECTRAN is geared to IBM mainframes and incorporates vector extensions to FORTRAN [Paul 1984].

W2—W2 is a programming language for the WARP architecture and other systolic arrays [Gross & Lamb 1986]. It combines elements of both Pascal and C. Users are allowed to specify message-send and message-receive operations between PEs.

XX—XX is geared to the CHiP research architecture [Snyder 1984]. Users can define independent processes, name data streams, and can use an explicit "idle" statement.

4.2.1.4 Debuggers for NvN Architectures

Throughout the process of developing NvN software many types of tools are needed to increase productivity, such as debuggers, monitors, and analyzers. Table 4-5 identifies debuggers used in conjunction with parallel architectures. These debuggers could possibly be integrated into a comprehensive programming support environment.

Table 4-5. Debuggers for NvN Architectures

Debugger	Literature Citation
Belvedere	[Hough, et.al. 1987]
Instant Replay	[LeBlanc, et.al. 1987]
Mdbx	[Symult 1988]
Pdbx	[Padua, et.al. 1987]

Belvedere—Belvedere is a trace-based debugging tool for message-passing architectures [Hough and Cuny 1987].

Instant Replay—Instant Replay is a debugging tool developed at the University of Rochester for the BBN Butterfly [Leblanc and Mellor-Crummey 1987]. Instant Replay regulates and records access to shared memory objects.

Mdbx–Mdbx is a source level debugger developed for the Symult 2010 computer [Symult 1988]. It is based on the standard UNIX dbx debugger allowing a programmer to debug a single process in the multi-processor environment.

Pdbx—The Pdbx tool, developed by Sequent Computer Systems, debugs multiple UNIX processes on Sequent shared memory architectures [Padua, Guarna and Lawrie 1987].

4.2.1.5 Performance Monitors for NvN Architectures

Table 4-6 identifies performance monitors used in conjunction with parallel architectures. These monitors could possibly be integrated into a comprehensive programming support environment providing a mechanism for determining an executable programs' overall performance or detecting performance bottlenecks.

Table 4-6. Performance Monitors for NvN Architectures

Performance Monitor	Literature Citation
HyperView Monit SeeCube	[Malony, et.al. 1988] [Kerola, et.al. 1987] [Couch 1987]

HyperView—HyperView is a performance visualization tool for distributed memory hypercube architectures [Malony and Reed 1988]. HyperView provides graphical displays of both system activity and performance statistics. HyperView incorporates some features of the antecedent SeeCube tool; however, it is based on the X-Windows environment and user interface libraries that are part of the Faust Environment at the University of Illinois CSRD.

Monit—Monit is a trace-based performance monitoring tool for parallel PPL (extended C) programs running on Sequent computers [Kerola 1987]. Monit provides graphical displays on Sun workstations.

SeeCube—The SeeCube tool provides trace-based graphical displays of application performance on hypercube architectures[Couch 1987, Couch 1988]. The tool executes in a Sun View windows environment.

4.2.1.6 Programming Models for NvN Architectures

In order to attain maximum performance on many NvN machines, programs need to be rewritten and/ or new algorithms need to be developed. Several researchers believe the most efficient method for designing application software is to use new programming models. Table 4-7 identifies programming models for NvN architectures.

Table 4-7. Programming Models for Parallel Computing

Programming Models	Literature Citation
Actors	[Agha 1986]
E-L	[Karr, et.al. 1989]
I/O Automata	[Lynch and Tuttle 1988]
Linda	[Ahuja, et.al. 1986]
Model	[Prywes, et.al. 1986]
Paralation	[Sabot 1988]
Unity	[Chandy and Misra 1988]

Actors—Actors [Agha 1986] provides a design approach to fine-grained parallel programming. The basic concepts of Actors were used as the basis for programming the Cosmic Cube in the Cantor language [Athas and Seitz 1988]. The Actors approach is based on message-passing as the means of communication between concurrent objects that consist of a code area and a small private memory. Objects respond to messages and can create new objects. References to an object that has been created but not yet instantiated constitute what is called "futures". Athas and Seitz used compile-time flow analysis of futures for load balancing and for preserving locality of references. Analysis of ratios of messages sent to messages received and of object creation patterns are retained and used as input to heuristic procedures that map objects to processors.

E-L—The key to providing a framework for dealing with the variety of software methodologies and hardware targets is an open architecture system which provides:

- a flexible linguistic medium, so that a user of the system can describe the domain of discourse

- an open-ended tool set, so that a user of the system can easily use existing tools and easily contribute new ones

- a solid base and principles of extensions, so that the user's contributions become part of a coherent whole.

Software Options, Inc., under DARPA support, has been developing a system called E-L (Environment and Language) that can be used as a platform for supporting many software techniques for non-von Neumann architectures and for integrating these techniques. Moreover, because of E-L's emphasis on program transformation, it would be an ideal basis for coping with multiple hardware targets. E-L provides the ability to state solutions in their most natural way, and also provides a way of expressing how these initial solutions are transformed to fit well on given hardware. In particular:

- It is straightforward to extend E-L's surface grammar to provide special-purpose notation that nevertheless looks quite built-in.

- Because many levels of the program coexist in base E-L, it is possible to fine-tune one aspect of a construct while letting other aspects be more general, and perhaps less efficient.

- One may supply special purpose transformation tools that take the place of what would ordinarily have to be done as an integral part of a compiler.

- The programmer may debug in the idiom of the extension, not of its implementation, because all of the translations and transformations specify how executions are linked.

The E-L approach is quite different from that of designing a new language or even extending an existing one. Rather than writing a new compiler or extending an existing one, the task is to build a few tools that transform "base E-L" (see below) in a certain way. The tools rely to some extent on transformations that can be stated declaratively, and are thus relatively easy to change. Each stage of the transformation process results in an executable base E-L program, an important factor in debugging the tools. Finally, the last stage of translation has primitives that are so close to hardware that extensions to an existing E-L code generator for the controlling computer are quite straightforward.

A central concern of E-L is that of extensibility. E-L provides a number of innovations in the language and environment areas to achieve this extensibility. Chief among these innovations is that E-L has multiple layers of language. There is a surface syntax in which programs are written and read that provides a format similar to many conventional programming languages except that spacing and indentation are used for grouping (like the more conventional blocks) and fonts are used for emphasis. Programs presented in the surface syntax are *reduced* (i.e., transformed) into equivalent programs in base E-L, a very simple language with a firm semantic basis. Tools such as analyzers, compilers, and debuggers operate on base E-L, but the naive user is generally unaware of base E-L and deals with programs in the surface syntax. The surface syntax is extensible in the sense that it can be augmented with additional grammar rules to provide notations appropriate for particular application domains. Programs written in extended notations are ultimately reduced to base E-L.

While a naive user is unaware of base E-L, an extender necessarily encounters base E-L in several ways. First, when extending surface E-L, the semantics of a new construct is given by indicating how it reduces to base E-L. Second, the analysis and transformation of programs is entirely the domain of base E-L.

Base E-L may be thought of as defining the syntax of a single construct, a *term*. There are two kinds of terms that are primitive, *names* and *constants*, and there are two kinds of terms that are made up of other terms, *abstractions* and *applications*. An abstraction allows the introduction of nomenclature; it is essentially the lambda expression of Common LISP , with some minor technical differences. An abstraction is written as:

- λ x [*term*, ..., *term*]

An application is written as in Common LISP , with the left parenthesis preceding the operator:

- (*term...term*)

Base E-L functions can take and yield varying numbers of values.

Although the syntax of base E-L is fixed, the primitives of base E-L are not. Consequently, the set of allowable transformations is also not fixed. This provides a controlled semantic extensibility that makes it possible for E-L to accommodate many different approaches to parallelism and concurrency.

Another important point is that functions, both primitives and those arising from the evaluation of abstractions, may be passed as arguments, returned as results and, quite significantly for the study of parallel and concurrent algorithms, stored in data structures. The reduction of flow-of-control surface E-L constructs invariably embeds part of the construct inside an abstraction, and applies a function to one or more function values in the base E-L version. For example, consider the iteration construct:

For *names* **in** *iteration* **do** *body*

The corresponding base E-L uses the function <u>iterate</u> to obtain generic behavior over the *iteration:*

(iterate *iteration'* λ*names'* [*body'*])

The primed versions of the pieces are the recursively reduced versions of the non-primed surface versions in the original. The role of iterate is to examine the value produced by *iteration'* and to call the abstraction repeatedly, supplying the proper values as operands. Each call corresponds to what the surface programmer thinks of as one iteration of the loop. There is no special machinery in base E-L to support iteration.

E-L has a notion of a *type-template*. Consider the following function header:

Function concatenate(x:list(?t), y:list(?t)) -> list(?t)

Because t is not fixed, it is not possible to say statically what type x or y has. On the other hand, this declaration at least says that x and y will always be lists and, moreover, in any given instantiation of concatenate, they will be lists of the same type and the result will also be a list of that same type. The type-template for concatenate would be written in surface E-L as:

template (t) (list(t), list(t)) -> (list(t))

There is also a notion of a type-template in base E-L. Such a value can be applied to operands to obtain a type, and it can be used in a unification-style algorithm: given a type-template and a type, it is

possible to obtain values for the parameters of the template that will produce that type, or to affirm that no such values exist. For example, given the type-template **template** (n, t) array (<n, n>, t), and the type array (<3, 3>, integer), the algorithm will produce 3 and integer. But if one of the "3's" is changed to "4", the algorithm will say that the desired values do not exist.

The reduction of surface constructs involving type-templates to base E-L is too involved to discuss here. Suffice it to say that in the base E-L for the body of concatenate, t is a parameter on an equal footing with x and y.

Function families.are another feature of E-L. A function family is a collection of functions that are, at surface E-L, referenced by the same name. The semantics of a function family is associated with the family and the members of the family are committed to realizing those semantics. A good example is the equality family, named "=". The semantics of equality is what you would expect— mathematically, an equivalence relation. The members of the equality family would test this property on the various data structures, such as integers, strings, arrays, and lists. The particular member of a family that is to be used in any context is determined by the type of its arguments. This determination is part of the job of the reduction mechanism that transforms surface E-L into base E-L. For a more complete account of E-L, see [Karr, et al. 1989], presented as a supplement to this report.

The breadth and depth of the facilities in E-L to support extension can be demonstrated by showing how a UNITY program can be directly embedded in E-L. This embedding does more than simply provide UNITY syntax and execution. It also provides UNITY program structuring facilities. The assumption "that there are no inconsistencies in definitions of variable, always sections, or initializations" in two programs being composed can be checked dynamically in those cases where it cannot be verified. A more complete discussion of this embedding can be found in [Karr, et al. 1989], a supplement to this Report.

Linda—Linda [Ahuja, *et.al.* 1986] is a model for parallelism that is conceptually very simple. A number of programming languages have been augmented with constructs from the Linda model.

The basic idea underlying Linda is that there is a tuple space, TS for short, that can be accessed by each of an arbitrary number of ongoing processes. The contents of TS is a collection of tuples of values. There are three constructs that are added to a programming language, **out, in** and **read**.

The **out** construct is of the form **out**$(v_1,...,v_n)$ and says that the tuple $<v_1,...,v_n>$ is added to TS.

The **in** construct is of the form **in**$(x_1,...,x_n)$ and says that if there is a tuple of length n in TS whose j-th component matches x_j for j = 1,...,n, then that tuple is removed from TS and the process containing the **in** continues execution. If there is no such tuple in TS, the process blocks until there is such a tuple. If more than one process is competing for the same tuple, one of them will receive it and the others will not (non-deterministically).

The **read** construct, **read**($x_1,...,x_n$), says that if there is a tuple of length n in TS whose j-th component matches x_j for j = 1,...,n, then that tuple is selected in TS and the process containing the **read** continues execution. If there is no such tuple in TS, the process blocks until there is such a tuple. The tuple selected is not removed from TS.

The basic idea of *matching* (simplified slightly) is as follows: If x_j is a value then the tuple identified must have that value in its j-th position. If x_j has the form var y_j and y_j has type T, then the tuple identified must have a value of type T in its j-th position and, on successful matching of all components, y_j is bound to the value in the j-th position of the identified tuple.

I/O Automata—I/O automata [Lynch and Tuttle 1988] provide a model for discrete event systems consisting of concurrently operating components. Such systems are characterized by the fact that they continuously receive input from and react to their environment.

Each system component is modeled as an I/O automaton which is essentially an [possibly infinite state] automaton with an action labeling each transition. A fundamental property of the model is that a distinction is made between those actions that are under the control of the automaton and those whose performance is under the control of the environment. An automaton's actions are classified as "input", "output", or "internal". An automaton generates internal and output actions autonomously, and transmits output instantaneously to its environment. In contrast, the automaton's input is generated by the environment and transmitted instantaneously to the automaton. An automaton is unable to block inputs transmitted to it (although it may, of course, disregard certain inputs that it receives).

Model—Model [Prywes, Shi, Szymanski and Tseng 1986] is a complete programming system employing three primary tools: a Configurator, the Model Compiler, and the Timer. High-level configuration specifications are input to Configurator and individual module specifications are input to the Model Compiler. The Compiler creates a module flowchart which is input to the Timer. The Configurator and Model Compiler arrange for appropriate data interfaces, and they attempt to optimize concurrency of the applications or components thereof.

Paralation—The Paralation (a contraction of PARALlel reLATION) model [Sabot 1988] is an abstract model that consists of two data structures and four carefully chosen operators. The Paralation model was explicitly designed to be an extension of another language. Moreover, an instance of such an extension has been made, to Common LISP.

One of the two data structures of the Paralation model is called a field. From one point of view, a field behaves very much like a one-dimensional array. However, from another point of view, a field is unlike an array because its elements may be thought of as residing in different memories among which there is a means of communication. A paralation is a collection of fields. Each paralation contains a special field called its index field and there is a **make-paralation** operator that creates a

new paralation with a single field, its index field, and returns that field. A good mental image for a paralation is given by Table 4-8.

Table 4-8. An Example of a Paralation

index	field$_1$	field$_2$	...	field$_k$
0	$d_{0,1}$	$d_{0,2}$	.	$d_{0,k}$
1	$d_{1,1}$	$d_{1,2}$		$d_{1,k}$
2	$d_{2,1}$	$d_{2,2}$	.	$d_{2,k}$
...	...	...	.	...
n-1	$d_{n-1,1}$	$d_{n-1,2}$		$d_{n-1,k}$

Each of the columns corresponds to a field. According to [Sabot 1988], a site is "the place where all values in a paralation that have the same index are stored". In other words, a site corresponds to a row in the above picture.

A paralation is a means for structuring fields and their elements that carefully defines field locality or nearness between field elements. It is worthy of note that [Sabot 1988] is quite explicit that a paralation is not a data object: there is no way to name a paralation and there are no operations defined on paralations.

There is an elementwise evaluation operator that allows programs to be evaluated independently in every site of a paralation to compute the elements of a new field in that paralation. This is how parallel computation is performed in the model.

Data can be moved between paralations as well as between fields in the same paralation. This involves the second of the paralation data structures, a *mapping*, which embodies a pattern of data movement from sites to sites of the paralation(s). A mapping is produced by the match_operator, whose arguments are source and destination fields of the paralation(s). A mapping is in turn an argument to a powerful parallel assignment statement. The data movement can be one-to-one, one-to-many, or many-to-one. In the latter case, a *combining* operator can be specified to resolve collisions; such a move thus involves computation as well as communication.

UNITY—UNITY [Chandy and Misra 1988] is a theory: a computational model (including notations for writing program specifications) and a proof system.

The form that a UNITY program (specification) takes is:

> **Program** *program-name*
> **declare** *declare-section*
> **always** *always-section*
> **initially** *initially-section*
> **assign** *assign-section*
> **end**

The interpretations of the various sections are:

- The *declare-section* names the variables used in the program and their types. The syntax is similar to that used in Pascal.

- The *always-section* is used to define certain variables as functions of others. It is not necessary but is convenient.

- The *initially-section* is used to define initial values of certain variables. Uninitialized variables are assumed to have arbitrary values initially. It is further assumed that the equations establishing the initial values of the variables are not circular.

- The *assign-section* contains a set of assignment statements. An assignment statement may be a simple assignment, a multiple assignment, or a set of simple and/or multiple assignments to be done in parallel. No variable can be given two distinct values as the result of a single assignment statement.

The execution of a program starts in a state where the values of variables are as specified in the *initially-section*. In each step, any one statement is executed. Statements are selected arbitrarily for execution, though it is required that in an infinite execution of the program each statement is executed infinitely often. A state of the program is called a fixed point if and only if execution of any statement in the program while in this state leaves the state unchanged. An example would be a program that sorted a finite array of integers. When the sort is completed, the program has reached a fixed point.

An example of a UNITY program exhibiting parallelism is the following sort program:

Program sort

...

assign

$<\|\ i : 1 \leq i \leq N \wedge even(i) ::$

$\quad A[i],A[i+1] := A[i+1],A[i]$ if $A[i] > A[i+1]>$

$<\|\ i: 1 \leq i \leq N \wedge odd(i) ::$

$\quad A[i],A[i+1] := A[i+1],A[i]$ if $A[i] > A[i+1]>$

end

Here there are two quantified-assignments separated by "," the selection operator; the interpretation is that on each cycle one of the two components is selected and the several individual assignments that are specified in the one chosen are carried out in parallel. By choosing the odd or the even elements one can increase the parallelism up to N because there are N/2 parallel statements, each with two assignments, that may be executed at a given step. Obviously, dealing with any subset of the elements of A that did not allow interference in the values being assigned to any element would also work. It is assumed that A and N have been appropriately declared and initialized.

4.2.1.7 Simulators for NvN Architectures

In the process of designing algorithms for NvN machines and designing new machines many performance questions arise that can be quickly resolved through simulation. Table 4-9 identifies simulators for NvN architectures.

Table 4-9. Matching Simulators to Literature Citations

Hardware Simulator	Literature Citation
B-HIVE	[Agrawal, et.al. 1986]
Nestor	[Nestor Inc.]
NETtalk	[Sejnowski, et.al. 1987]
NeuroSoft	[Hecht-Nielson Neurocoputers]
PAW	[Melamad, et.al. 1985]
SIMON	[Fujimoto 1983]

B-HIVE Graph Mapping Tool—This graph mapping tool was developed at North Carolina State University as part of the B-HIVE project [Agrawal, Janakiram, and Pathak 1986]. The tool maps a computational flow graph (representing a program) onto a computer resource graph (representing an architectural configuration) in a manner that seeks to minimize both computational and communication time. Simulation software gathers statistics on average communication distance between sending and receiving nodes, average channel utilization, etc.

Nestor—Nestor, Inc. offers their Nestor Development System for developing neural network applications.

NETtalk—NETtalk is a VAX-based simulator for neural networks reported in [Sejnowski and Rosenberg 1987].

Neurosoft—Hecht-Nielsen Neurocomputers offers their Neurosoft product for developing neural network applications. They also offers the Anza-Plus Neurocomputing Coprocessor to speed the learning and execution for application development.

PAW—PAW (Performance Analysis Workstation) is a modeling tool develoned at Bell Laboratories [Melamed and Morris 1984]. PAW employs a queueing network model c nsisting of node topology, transactions within nodes, and the dynamic flow of transactions between nodes. The tool produces performance statistics and graphical displays. It runs under UNIX on a Teletype Dot-Mapped Display 5620 terminal as part of the AT&T UNIX-Toolchest. PAW has three major components: a graphics editor, a text editor, and a simulator.

SIMON—SIMON simulates the behavior of parallel C programs [Fujimoto 1983, Heller 1984], but it lacks built-in capabilities for modeling Interconnection Network topologies [Nichols and Erdmark 1988].

4.2.1.8 Software Tool Sets for NvN Architectures

A few research institutions are working on providing a tool sets that could be considered a programming environment for NvN machines. Table 4-10 identifies ten of these existing software development tool sets for parallel architectures.

Table 4-10. Matching Tool Sets to Literature Citations

Tool Set	Literature Citation
CODE UCG	[Browne, et.al. 1989]
CODE	[Browne, et.al. 1989]
Faust	[Padua, et.al. 1987]
Implementation Assistant	[Segall, et.al. 1985]
IR	[Smith et.al. 1988]
MPF	[Maloney 1987]
PARET	[Nichols, et.al. 1988]
PARSE	[Casvant, et.al. 1987]
PIE	[Segall 1985]
PISCES 2	[Pratt 1987]
POKER	[Snyder 1984]
Rn	[Carle, et.al. 1986]
SCHEDULE	[dongarra, et.al. 1986]
VERDI	[Shen 1988]

CODE UCG Tool—The CODE UCG (Unified Computation Graph) tool was developed at the University of Texas, Austin [Browne, Azam and Sobek 1989]. It is an interactive tool that allows users to specify parallel programs for MIMD architectures in terms of schedulable units of computation and an "extended directed graph" that specifies dependency relationships. Users can specify data, demand, mutual exclusion and control dependencies that determine whether computational units are executed. The specification of dependency relationships and computational unit source code contents are carefully separated operations, driven by interactive menus and "form" (template) construction. The output of this construction process is transformed into an architecturally independent program specification that incorporates graph and template information. This architecture-independent specification is then processed by a translator that is language-specific and has knowledge of the execution environment.

CODE—The Computationally Oriented Display Environment (CODE) developed at the University of Texas, Austin provides a graphical programming environment. Programming is accomplished by creating a generalized dependency graph and defining the nodes and arcs. A node is defined by providing the source code for a procedure written in a high level language (e.g., FORTRAN, C, or Pascal). An arc definition represents either a data dependency, demand dependency, mutual exclusion dependency, or control dependency.

Faust—Faust is a software engineering environment for scientific computing that is being developed at the University of Illinois; it is targeted to the integration of several software development tools through a window-based interface. Faust can automatically create a subroutine call graph from source code. Faust also supports other details, such as process graphs and data dependency graphs. Faust incorporates the Parafrase II restructuring tool into its environment.

Implementation Assistant—The Implementation Assistant tool is part of the Carnegie-Mellon University PIE environment [Segall 1985]. This tool predicts parallel program performance, aids users in selecting implementation paradigms, and generates parallel programs in a semi-automatic manner. The implementation paradigms handled by the tool include master/slave, recursive master/slave, heap organized problem, pipeline, and systolic multidimensional pipeline.

IR—The IR programming environment being developed at the Georgia Institute of Technology focuses on developing, debugging, and optimizing large programs. The proposed toolkit would provide the following functions:

1. A parallelizer that would interactively examine a user's source program and suggest modifications to either increase parallelism or improve parallelism introduced by the user (e.g., eliminating non-portable or inefficient use of parallel constructs) [Smith, et.al. 1988].

2. A static analyzer that would interactively simulate the execution of a user's source program, to locate potential bugs or anomalies caused by the concurrent execution of tasks [Applebee, et.al. 1985].

3. A dynamic debugger that would interactively execute a user's source program.

The toolkit is based upon the premises that the source language is FORTRAN, with concurrency construct extensions, and the users will use the toolkit in an interactive edit, compile, analyze/test cycle, until the program is ready for production use. The toolkit is intended for General Purpose, Multiple-PE, specifically shared-memory, and is supposed to be portable to help programmers adapt their programs to new architectures.

MMS—The Multiprocessor Monitoring Systems tool environment (MMS) contains tools for debugging, performance analysis and visualization of multiprocessors and their program execution.

Apart from the functionality of the tools, MMS offers portability to various parallel architectures, expandability, and adaptability with new tools and languages, and support of several abstraction levels. The main design concept of MMS is a hierarchical layered model for tool environments [reference]. MMS has been under development at the Technical University of Munich since summer 1987. The first tool of the environment, the concurrent debugger, has been completed. The performance analyzer and the visualization tools are in the specification phase. Currently, the tool environment is adapted to target systems consisting of more than one processor element. A 32-node iPSC from Intel is the first target system for the multiprocessor implementation of MMS.

PARET—PARET (Parallel Architecture Research and Evaluation Tool) is an interactive graphical tool developed by Bell Laboratories [Nichols and Edmark 1988]. PARET models multicomputer system performance using data flow graphs to represent processes, links, and buffers. PARET makes use of behavioral simulation, which is controlled by a discrete event-driven simulator.

PARSE—PARSE is designed to be a software environment for reconfigurable non-shared memory machines. It consists of a collection of language interfaces and debugging and analysis tools. PARSE is designed to improve programmer productivity, where productivity is characterized by three factors: reducing development time, improving performance and efficiency, and improving reliability. One tool which is always used by PARSE is XPC (eXplicitly Parallel C). Every program is eventually expressed as XPC code. XPC is designed to provide explicit parallel control, data allocation, and program-controlled machine reconfiguration.

PCT—The PCT (Program Constructor Tool) is part of the PIE environment at Carnegie-Mellon University [Segall 1985]. PCT provides, as the primary user-interface, an environment metalanguage MP. PCT's components include an MP editor, a status and reference monitor, and a relational representation scheme. PCT transforms development and run-time statistics to a form that can be stored as part of an intermediate program representation. The monitoring component of PCT uses source code insertions to oversee monitoring and debugging operations.

PISCES 2—PISCES 2 is an environment for programming parallel machines. It was designed to provide an efficient execution environment for scientific and engineering applications on a variety of Class VI NvN architectures. PISCES 2 relies on FORTRAN 77 and UNIX as the underlying sequential language and operating system, respectively. PISCES 2 was developed by Dr. T. Pratt at the University of Virginia. PISCES FORTRAN is a parallel programming language for the Flex/32 [Pratt 1987]. PISCES FORTRAN language extensions include message-passing facilities, tasks, shared common blocks, "forces" for medium grain parallelism (code segments and loop iterations), locks and critical sections, and "windows" into array slices.

Rn—Rn is a program development environment developed at Rice University, and provides for the development of whole programs rather than individual source modules. Rn provides a language sensitive editor for entering source code, a module editor for composing collections of modules, a compiler that optimizes whole programs, and an interpretive debugger. All the pieces of the Rn environment utilize a single database that contains an intermediate representation of the program. Rn incorporates the PFC restructuring tool in its kit.

SCHEDULE—SCHEDULE is a package of routines that provides an interface between FORTRAN programs and a parallel machine. The FORTRAN routines communicate through shared variables. A programmer defines dependency relations between routines (via SCHEDULE calls), and SCHEDULE maps the program onto the hardware. SCHEDULE is designed to be a portable environment for developing parallel FORTRAN programs. Existing FORTRAN subroutines can be called through SCHEDULE without modification. Thus, users have access to a large body of existing library software. Machine intrinsics are invoked by SCHEDULE. Users are relieved of modifying code that is transported from one machine to another. SCHEDULE is currently running on VAX, Alliant, and Cray-2 computers.

VERDI—VERDI (Visual Environment for Raddle Design and Investigation) is being developed by the Microelectronics and Computer Consortium [Shen 1988]. The tool facilitates the development of distributed systems with the Raddle design language by providing graphics facilities for design specification, language-sensitive editing for computation and variable specification, and display capabilities for performance monitoring.

4.2.2 Analysis of Software Tools for NvN Architecture Classes

This section correlates the NvN architecture classes defined in Chapter II and the software development tools identified in Section 4.2.1. In addition, our analysis of software tools provides a discussion of existing tools and recommended tools for each architecture class. Table 4-11 correlates architecture classes and functionality of identified tools. Clearly the largest efforts have been directed toward the Class I and Class IV architectures, which corresponds to the number of available machines per class.

Table 4-11. NvN Architectures and Identified Software Tools

	Pipelined Vector Uniprocessors	Rhythmic Cellular Control	Processor Arrays	Associative Memory Processors	Operand-Driven	GPMPE	Neural Networks
Operating Systems	X			X		X	
Optimizing Compilers	X	X	X	X	X	X	
Programming Languages	X	X	X	X	X	X	
Debuggers	X	X	X	X		X	
Performance Monitors	X					X	
Programming Models	X	X	X		X	X	
Hardware Simulators	X	X	X		X	X	X

4.2.2.1 Software Tools for NvNACS Class I: Pipelined Vector Uniprocessors

4.2.2.1.1. Analysis of Existing Software Tools and Techniques

The most commonly available software tools for these architectures consist of restructuring compilers and libraries of mathematical subroutines. Such subroutine libraries for vector applications are often extensive. Table 4-12 shows examples of existing tools for Class I machines.

Table 4-12. Examples of Tools for Pipelined Vector Uniprocessors

Programming Languages	Optimizing Compilers	Software Tool Sets
Vector C FORTRAN 8X	KAP Parafrase VAST	PFC PTOOL Rn

Programming languages for application development on these architectures are typically standard HOLs (e.g., FORTRAN, Pascal) with language extensions or compiler directives identifying vector data types or operations. Restructuring software tools and compilers, such as PTOOL and PFC, are capable of transforming standard HOLs into efficient executable code through sophisticated data dependency analysis.

To a large extent machines in this architecture class benefit from tool development for standard von Neumann computers. Many common tools, such as editors, operating systems, debuggers, performance monitors, and programming models, can be effectively used to develop software for these machines.

4.2.2.1.2. Analysis of Needed Software Tools for NvNACS Class I

Software tools available for the Pipelined Vector Uniprocessor class, primarily vectorizing compilers, are probably the most highly developed tools extant for NvN architectures. However, software tools that support the design of application programs are not nearly so prevalent. Interactive tools (e.g., PTOOL) that identify language constructs that prevent vectorization support the development of optimal source codes are a significant step forward toward providing support for the design process.

Currently available tools lack design aids that would allow users to analyze performance implications of program attributes. Therefore, a useful tool for Class I architectures would determine the impact on performance of attributes such as vector operand lengths. One possible approach is to model the effect of alternative program structures on pipelining efficiency.

A tool having the ability to model a variety of existing vector uniprocessors would benefit application developers. Such a tool might use a comprehensive parameterization scheme for specific architectural features (e.g., pipeline size). Forthcoming availability will depend heavily on how quickly the computer sciences research community can define appropriate representation schemas for different architectural features.

4.2.2.2 Software Tools for NvNACS Class II: Rhythmic Cellular Control

4.2.2.2.1. Analysis of Existing Software Tools and Techniques

Existing tools supporting the development of application programs for systolic and wavefront array architectures consist primarily of special purpose programming languages (e.g., W2) and proposed techniques for mapping graph theoretic algorithms that represent data flows to the features of Class II machines. Such techniques are discussed by several authors (e.g., [S-Y. Kung 1987, Navarro, et al., 1987]). These proposed techniques have been used in research environments. Table 4-13 presents available tools for this class.

Table 4-13. Examples of Tools for Class II Machines

Optimizing Compilers	Programming Languages
Saxpy Matrix-1 FORTRAN	AL APPLY MDFL W2

4.2.2.2.2. Analysis of Needed Software Tools for NvNACS Class II

The primary tool capability needed for this architectural class is a facility for mapping algorithms to existing systolic and wavefront architectures. This will likely involve a composition facility for creating graph representations of algorithms, as well as a means for mapping algorithms to actual machines. Tool capabilities should include:

- comparing alternative architecture topologies
- performance modeling
- mapping algorithms to machines that have fewer PEs than algorithm calculations/ accumulations (e.g., matrix size/PE-count mismatch).

Developing fault-tolerant applications would require modeling the effects of swapping in replacement PEs, particularly to determine synchronization impacts.

Design tools are needed for rhythmic cellular control architectures. A design tool would help decompose problems to match a machine's communications network. In addition, monitoring tools are needed; they would require cooperating software components on both host and target architectures.

4.2.2.3 Software Tools for NvNACS Class III: Processor Arrays

4.2.2.3.1. Analysis of Existing Software Tools and Techniques

Programming languages are the predominant form of existing software tools supporting the development of applications for Processor Array architectures. Language features and extensions for SIMD architectures of this type usually involve array-structured operands and often mirror the particulars of the Interconnection Network connecting the processors (e.g., mesh, bit-plane organization). Table 4-14 presents available tools for Class III architectures.

Table 4-14. Examples of Tools for Processor Arrays

Programming Languages	Operating Systems
Actus II BLAZE Glypnir MLP Parallel Pascal	TOS

4.2.2.3.2. Analysis of Needed Software Tools for NvNACS Class III

There are no tools that facilitate determining which of several candidate processor arrays is best suited to a particular application. However, it is unlikely that any generic tool would ever be constructed because of the extreme structural dissimilarity of processor arrays (e.g., Connection Machine, MPP) and the fact that some are targeted to a particular application (e.g., the Motorola T-ASP for signal processing).

Two categories of software development tools are needed to support processor array class machines. First, tools are needed that facilitate comparisons among machine features to ascertain the suitability of different PE interconnection topologies such as bit-plane organization (e.g., MPP, DAP, CLIP4) or various mesh schemes. Second, there is a need for algorithm partitioning tools to support the efficient mapping of algorithms to machines having fewer than the ideal number of PEs.

4.2.2.4 Software Tools for NvNACS Class IV: Associative Memory Processors

4.2.2.4.1. Analysis of Existing Software Tools and Techniques

Over the past decade, much of the interest in content-addressable memory has been shifted to neural network research. Therefore, current developmental work on support software for associative memory processors is essentially restricted to just a few manufacturers (e.g., Loral Systems for the STARAN and ASPRO machines). Existing support software includes operating systems, compilers, and expert system development tools. Table 4-15 presents tools available for Class IV architectures.

4-15. Examples of Tools for Associative Memory Processors

Operating System	Compiler
Unix-ASPRO	FORTRAN C

4.2.2.4.2. Analysis of Needed Software Tools for NvNACS Class IV

Since the natural strength of these architectures is parallel database matching operations (e.g., correlating sensor-events with platforms), two practical questions arise in determining associative memory architecture applicability:

- are they superior to other architectures in such matching functions?

- can they perform other than matching functions rapidly enough to be useful as general purpose machines?

Software performance modeling tools could provide appropriate answers to both questions.

In addition, software design aids, such as algorithm partitioning tools, would be of considerable help in developing applications for associative memory machines, because of the lack of widespread programmer experience in designing efficient algorithms for these architectures. The support software provided by the Loral Systems Group for developing expert systems on the ASPRO effectively addresses this problem [Lott 1987].

4.2.2.5 Software Tools for NvNACS Class V: Operand-Driven

4.2.2.5.1. Analysis of Existing Software Tools and Techniques

Data Flow and Reduction Machine architecture research is intertwined with research into data flow languages and programming models. Most operand-driven machines are university computer

research center prototypes; several data flow programming languages for these machines have been proposed and prototype compilers are being developed. However, there seems to be no mature software development support tools currently available for these architectures. Table 4-16 presents tools available for Class V architectures.

Table 4-16. Examples of Tools for Operand Driven

Programming Languages
ID VAL

4.2.2.5.2. Analysis of Needed Software Tools for NvNACS Class V

Research in operand-driven architectures is relatively immature; consequently, there are very few software development tools that could be compared against generic requirements for the class. Two kinds of generic tools are crucial for further advances in this area. First, tools are needed for top-down design of data flow algorithms and for transforming standard HOL source code constructs into data flow constructs, because there is no widespread familiarity with the data flow programming paradigm. Second, trace-based performance modeling tools could help in identifying applications that can effectively exploit data flow and reduction machine architectures.

4.2.2.6 Software Tools for NvNACS Class VI: General-Purpose, Multiple-PE

4.2.2.6.1. Analysis of Existing Software Tools and Techniques

Recent research activities have yielded a significant number of software support tools for GPMPE architectures, including development support tools, languages, environments, and operating systems. Table 4-17 presents tools available for Class VI architectures. Although some existing tools are geared to both private and shared memory architectures (e.g., the University of Texas–Austin CODE environment and tools [Browne, et al.. 1989]), most of the tools are geared to only one such category. Because of these different orientations, the existing software tools for each type of memory architecture will be discussed separately in the following paragraphs.

Table 4-17. Examples of Tools for GPMPE

Operating Systems	Optimizing Compilers	Programming Languages	Debuggers	Performance Monitors	Programming Models	Hardware Simulators
Chrysalis COS MACH MMOS PASMOS UNIX	ART Parafrase II PAT Sigma	BLAZE Cantor Multi-Pascal Occam XX	Belvedere Instant Replay PCT Pdbx	HyperView Monit SeeCube	I/O Automata LINDA MODEL	CODE UCG CODE PARET PAW

Sophisticated application design and partitioning tools have been developed for private memory GPMPE architectures; some provide advanced performance modeling capabilities (e.g., PARET and B-HIVE). Mature debugging and performance monitoring tools have also been constructed.

Application design support tools for shared memory GPMPE architectures are not as plentiful as for private memory machines. Segall's Implementation Assistant (IA) tool [Segall 1985], which automates the comparison of candidate algorithm control paradigms, is perhaps the most ambitious design tool effort. In addition, several debugging and performance monitoring tools have been implemented; most are based on UNIX or UNIX-like operating systems. Although advanced tools for source code analysis and transformation exist, many of these tools emphasize vectorization for multiple-cpu vector architectures, such as the Cray X/MP (e.g., ART, PAT), and have capabilities similar to the tools developed for pipelined vector uniprocessor architectures (e.g., PFC, PTOOL). Parafrase II, another code analysis tool, seems to be geared to both vector and non-vector oriented shared-memory GPMPE architectures.

4.2.2.6.2. Analysis of Needed Software Tools for NvNACS Class VI

Although tools and environments have been constructed that provide sophisticated NvN software development capabilities, this survey has not identified any integrated set of capabilities that unites algorithm specification and performance modeling feedback with interactive assistance on effective design principles. There appear to be no design tools available that aid in predicting the comparative performance of a given algorithm on a variety of GPMPE architectures; this is in stark contrast to such code analysis tools for both single and multiple CPU vector architectures (e.g. PTOOL and PAT).

Alternative modeling capabilities for alternative private memory architecture topologies (e.g., B-HIVE graph tool and PARET) do not appear to be matched with similar modeling capabilities for shared memo. architectures.

A well-integrated set of capabilities are needed, including (1) facilities for specifying algorithms, (2) modeling an algorithm's implementation on alternative GPMPE architectures, and (3) obtaining interactive "advice" on effective design principles for specific NvN architectures.

Designers faced with selecting the best parallel architecture for a specific application need modeling tools for predicting application performance on both private and shared memory architectures. The University of Texas–Austin CODE UCG Tool, which allows application specification for both private and shared memory architectures, might prove useful. Performance modeling tools are needed for alternative shared memory architectures to help determine the performance effects of alternative mechanisms for memory access locking, of different memory access patterns, of various cache coherency strategies, and of alternative Interconnection Network technologies.

There remains a need for additional source code analysis tools that partition algorithms on private memory GPMPE architectures. Existing tools, such as Parafrase II, SIGMA (for Single Address

Space Architectures), and the Cantor compiler (which uses compile-time heuristics for hypercube architecture load balancing) could prove useful.

Performance modeling tools are needed for alternative shared memory architectures to help determine the performance effects of alternative mechanisms for memory access locking, of different memory access patterns, of various cache coherency strategies, and of alternative interconnection network topologies.

4.2.2.7 Software Tools for NvNACS Class VII: Neural Networks

4.2.2.7.1. Analysis of Existing Software Tools and Techniques

A significant amount of software exists to model the performance of proposed neural network applications predicated on paradigms such as Adaptive Resonance Theory, Back-propagation, Counter- propagation, and Competitive Learning.

In addition, software tools such as Hecht-Nielsen Neurocomputers Neurosoft product and Nestor Inc.'s Nestor Development System, are available for the creation of neural network software models. Table 4-18 presents tools available for Class VII architectures.

Table 4-18. Examples of Tools for Neural Networks

Hardware Simulators
Anza-Plus Neurosoft Nestor

4.2.2.7.2. Analysis of Needed Software Tools for NvNACS Class VII

Current software tools for modeling the performance of neural network solutions for various applications typically emphasize performance results in terms of the percentage of correct classifications, although the speed for obtaining results is often reported as well. The technical literature does not clearly indicate the existence of modeling software that can report on both aspects of performance for neural networks that are organized either for alternative learning or for recognition paradigms.

Software tools for evaluating the comparative suitability of various neural network learning paradigms (e.g., Back-propagation, Kohonen learning) would likely prove useful. However, most proposed neural network hardware architectures are radically dissimilar from other NvN architectures, due to the notion of embodying algorithms in inter-PE connections as well as in computations. It is difficult, therefore, to propose software tools that would compare the suitability of neural networks with respect to other Non-von Neumann architectures.

4.2.2.7.3. Examples of Software for Neural Network Architectures

    a. Neural Network Application Simulation

    (1)   NETtalk is a VAX-based simulator reported in [Sejnowski and Rosenberg 1987].

    (2)   The Globular Protein Secondary Structure Predictor is a Ridge 32-based simulator reported in [Qian and Sejnowski 1988].

    (3)   The Sonar Target Classifier is a Ridge 32-based simulator reported in [Gorman and Sejnowski 1988].

    b. Application-Independent Neural Network Modeling Software

    (1)   Hecht-Nielsen Neurocomputers offers their Neurosoft product.
    (2)   Nestor, Inc. offers their Nestor Development System.

    c. Neural Network Emulation Coprocessors

    (1)   Hecht-Nielsen offers their Anza-Plus Neurocomputing Coprocessor.

    (2)   The Helsinki University of Technology offers the Neurocomputers Neural Phonetic Typewriter, which was developed by Professor Kohonen.

4.2.3 Analysis of Existing Software Tools for Supporting a Life-cycle on NvN Architecture

Software tools exist which support each phase of a life-cycle as shown in Table 4-19. However, their effectiveness is questionable and will be discussed in the following paragraphs.

Requirements Analysis—Tools that exist in this phase of a life-cycle do not provide enough assistance to problem analysis for architecture selection. Performance constraints, problem definition and architecture characteristics direct the suitability of specific architecture types. This information, if available, is critical to design decisions. In the survey no tools were found to significantly aid in this part of the analysis.

Design—For the design phase tools were provided but have major deficiencies. Tools that address portability issues produce designs that are not optimal. The architecture specific tools produce more optimal designs but are dependent on subsequent phases in the life-cycle such as implementation and test which feed back information on performance. This shows a weakness in the current state of design tools and a bias towards life-cycle methodologies which iterate between the design, implementation and test phases.

Table 4-19. Tools Available for Each Phase of the Life Cycle

	REQUIREMENTS ANALYSIS	DESIGN	IMPLEMENTATION (CODE & DEBUG)	TEST	MAINTENANCE
Operating Systems			X		X
Optimizing Compilers			X		
Programming Languages			X		
Debuggers			X		
Performance Monitors			X	X	
Programming Models	X	X	X		
Hardware Simulators	X	X	X		
Tool Sets	X	X	X	X	X

Coding and unit testing—Most of the tools provided in this phase are architecture specific; automated code decomposition is still a research area; test and debugging techniques for NvN are similar to those used for TvN architectures and are ineffective for multi-processor/multi-memory systems.

Integration and test—The major deficiency in this phase is in the area of performance monitoring. For NvN machines, there are no standard performance metrics. Tools concentrate on machine efficiency rather that problem solution bottlenecks.

Maintenance—Rehosting tools have both performance and correctness problems.

4.3 THE AUTOMATION OF SOFTWARE DEVELOPMENT FOR NvN ARCHITECTURES

The analysis presented in this chapter of the report clearly indicates that the state-of-the art in software engineering for NvN architectures requires further research and investigation. This section discusses software development for NvN architectures in the future.

As the data clearly indicates NvN architectures are rapidly becoming computation engines for scientific and engineering research and development. Commercially available tools for these machines consist mainly of operating systems, language compilers and debuggers. Research communities are investigating tools that cross machine architectures, such as graphics based

programming tools, programming models, and new languages. While most of the research in these areas is interesting, few commercial quality products have emerged. Further research is needed in these areas and as that research proves fruitful, organizations need to build quality tools around those products.

Researchers have analyzed many of the issues related to coding and unit testing and efficiency of execution. Issues, such as load balancing, PE-to-PE communications, PE-to-Memory bandwidth, and replication of computation and data. These issues are being analyzed through research in the areas of operating systems and compiler optimizations. Many of the issues related to design and maintenance, such as reliability, portability, and readability, are being looked at by programming language designers and researchers creating programming models. While many of the specific issues are being addressed it would be useful to have tools that assist programmers resolve problems in these areas independent of other, larger functions.

Few research efforts are investigating phases of the life cycle other than the code and unit test phase. The survey identified no tools, which specifically address requirements analysis, design, integration and test, deployment, and maintenance. The initial phases of requirements analysis and and design are probably the most important areas where further research is needed. Early identification of NvN architectures can only assist in the development of efficient algorithms and software implementations for a project.

As the field of NvN computing matures, techniques and methodologies for software engineering will evolve. As these methodologies evolve inter-phase tools will emerge that assist in moving from requirements analysis to maintenance. As stated earlier documentation and traceability are not part of the life cycle phases, however, the need for communication and correctness for large projects most surely requires such tools. Automation between phases will potentially reduce the errors from inter-phase transitions.

Once methodologies are developed, software development environments can be created that automate the entire life cycle based on a methodology. Tools can be integrated that provide a consistent view of a project from requirements analysis through deployment and maintenance.

CHAPTER V. CONCLUSIONS

5.1 CONCLUSIONS ABOUT THE CURRENT STATE-OF-THE-ART

There are several instances of NvN machines for each of the seven classes defined in the NvN Architecture Classification Scheme (NvNACS). The majority of extant machines having NvN architectures are in Class I (Pipelined Vector Uniprocessor) and in Class VI (General Purpose, Multiple-Processing Element).

Many existing problems will be re-hosted on NvN machines, and there are an even larger number of remaining problems for which solutions will be sought using NvN machines. Because examples of each class are available in the marketplace, future utilization patterns will be defined by users' applications.

Many research efforts are focusing on the creation and implementation of efficient algorithms for NvN architectures. Many software tools are emerging from research institutions that address issues pertinent to efficient run-time implementations of algorithms, such as distributed operating systems, restructuring compilers, debuggers and performance monitoring tools.

There are a few efforts focusing on the software engineering issue of portability. Programming models such as LINDA, Actors and Model are examples of early efforts to resolve portability issues, as well as other critical issues, such as efficiency, reliability and maintainability.

5.2 RECOMMENDATIONS FOR ADVANCING THE STATE-OF-THE-ART

The rapid evolution of NvN architectures will continue to outpace the ability of programmers to utilize them efficiently for solving problems. The focus of research should be shifted toward software tools that support each phase of the life cycle.

Algorithm selection, architecture selection, load balancing, PE-to-PE and PE-to-Memory communications, and performance evaluation are critical issues, which need to be considered in each phase of the life cycle and require tools to deal with them. In addition, the software engineering issues of reliability, maintainability, portability, efficiency and productivity need to be addressed.

Many existing tools should be integrated into an initial environment focusing on the implementation of software. As new tools addressing other life cycle phases become available, they can be incorporated to create a robust programming environment.

New software engineering methodologies will emerge as many of the critical issues pertinent to NvN computing become better understood. Most likely these new methodologies will follow other current trends in software engineering and incorporate some type of iterative process, which combines the phases of the life cycle. These methodologies will evolve as NvN computing environments mature. Research needs to be conducted in the areas of requirements analysis and design to speed the evolutionary process.

APPENDIX A: ARCHITECTURE ASSESSMENT SKETCHES

This appendix to the Final Report contains concise, informal sketches of architecture assessments, including remarks on strengths, weaknesses, and likely application domains for selected categories in the Non-Von Neumann Architecture Classification System (NvNACS). Assessments are offered for at least each high-level class of architecture. In addition, assessments are provided for lower-level classification categories of the General Purpose Multiple PE Class.

This appendix does not constitute an exhaustive analysis of the strengths and weaknesses of major computer architecture categories.

A.1 Pipelined Vector Uniprocessor Architectures

A.1.1 Strengths

These architectures perform very efficiently for applications in which data is naturally represented as vectors and matrices.

Performance is best for applications that:

* have enough sequential numeric calculations to keep the pipeline(s) filled;

* use vectors with lengths that are equal to the number of vector registers, are a multiple of the vector register count, or are a factor of the register count (e.g., vector lengths of 4 or 8 for 16 registers).

Very sophisticated compilers that automatically vectorize application code exist for most of these architectures.

A.1.2 Weaknesses

Parallel execution of programming constructs that do not involve vectors is limited to pipelining and, possibly, use of multiple functional units.

Potential parallelism at the task or subroutine level is not exploited.

The performance of these architectures on symbolic processing or data base-oriented applications can be expected to vary considerably.

Software developers may be required to have detailed knowledge of how associated compilers vectorize code in order to construct applications that effectively exploit these architectures' strengths.

A.1.3 Application Analysis

The vector computation capabilities of these architectures are most often exploited for scientific and engineering applications, particularly modeling. Example applications surveyed include fluid dynamics, seismic modeling, and Navier-Stokes equations.

A.2 Rhythmic Cellular Control Architectures

A.2.1 Strengths

Individual PEs can be simple and inexpensive because each performs a very limited function. Fixed systolic and wavefront architectures are very efficient because of the high degree of parallelism achieved.

A.2.2 Weaknesses

The application-specific character of fixed systolic and wavefront architectures requires a new hardware configuration for each application, resulting in possibly non-trivial development costs, despite use of common modular components.

Developing high-level language applications for programmable systolic architectures may be difficult, since explicitly programming the necessary time delays is not supported by many standard high-level programming languages.

The systolic architecture practice of synchronizing all PEs with a global clock effectively limits the number of PEs (because of skewing that results from clock signals having to travel too great a distance to the farthest PEs [Kung 1984]).

A.2.3 Application Analysis

The most prevalent use of systolic and wavefront architectures is to perform matrix operations for signal processing applications. Academic researchers, however, have experimented with systolic architectures for text manipulation, data base searching, and automata implementation [Kung 1982]. Surveyed applications included radar, sonar, nucleic acid sequence comparison, and linear algebra.

A.3 Processor Array Architectures

A.3.1 Strengths

Potentially, a very large number of processors can be brought to bear on a problem, providing a high degree of parallel execution.

Processor array architectures are especially effective for applications that involve identical computations being performed on different data.

The synchronization scheme used is straightforward (broadcasting a single instruction that all PEs execute in lockstep), thereby eliminating the need for explicitly stating complex synchronization mechanisms in application software.

A.3.2 Weaknesses

Limiting parallelism to the instruction level, as these architectures do, precludes effective parallel execution at the task or procedure level. This has historically restricted the applicability of these architectures in some significant problem domains. However, architectures such as the Connection Machine, with 1-bit PEs that can essentially modify the broadcast instruction, appear to be applicable to domains previously ill-suited to array processor organization.

A.3.3 Application Analysis

Processor array architectures are commonly used for scientific and engineering applications similar to those often found on vector processor architectures. In addition, bit-plane oriented processor array architectures are particularly suitable for image processing applications. Surveyed applications included:

> satellite imagery and data processing
> numerical analysis
> Monte Carlo simulation
> partial differential equations solution
> weather forecasting
> nuclear energy modeling
> seismic data processing
> structural analysis
> passive and active sonar signal processing
> economic simulation.

A.4 Associative Processor Architectures

A.4.1 Strengths

The primary strength of architectures built around an associative memory is the speed with which highly parallel data search and comparison operations can be performed.

These architectures should be particularly effective for embedded military applications that require rapid matching of data base contents against records constructed from sensor-based data (e.g., match a record describing a radar emitter in the environment against stored records containing known emitters' pulse width, pulse repetition interval, etc.).

A.4.2 Weaknesses

These architectures' potential weakness for some applications is of the time required to load or reload the associative memory.

This could be particularly acute if the memory were too small to hold the relevant data base.

Associative memory processing architectures may not be flexible enough to efficiently implement a broad variety of applications and algorithms.

A.4.3 Application Analysis

Surveyed applications for associative processor architectures include tracking and surveillance, image and signal processing, and cartography.

A.5 Operand-Driven Architectures

A.5.1 Strengths

These architectures potentially provide a high degree of parallel execution at the instruction level.

Since these architectures appear to be especially promising for application domains and programming languages dominated by expression evaluation, they are likely to prove effective for rule-driven expert systems and for artificial intelligence applications (e.g., breadth-first searches of solution spaces).

A.5.2 Weaknesses

Few of the existing architectures of this class appear to handle PE failures effectively [Srini 1986].

A possible practical weakness results from the reliance of operand-driven architectures on applications being represented in largely experimental data-flow or functional programming languages. Since it is unlikely that there is widespread programming expertise in this area or mature software tools to develop applications in these forms, utilizing these architectures for large DoD programs seems unlikely in the near future.

A.5.3 Application Analysis

Most of the surveyed applications are intended to prove either the viability of functional and data-flow programming languages or of this kind of architecture [Trealeven 1982]. Therefore, it seems premature to characterize these architectures in terms of relevant application domains.

A.6 General-Purpose Multiple-PE (GPMPE) Architectures

The architectural characteristics of machines in this class are so diverse that any assessment of strengths, weaknesses, and surveyed applications for the class as a whole would not be useful. Instead, such assessments are given for appropriate subcategories of this high-level class.

A.6.1 Hypercube Topology, PE-to-PE Communication Architectures

A.6.1.1 Strengths

A significant advantage of hypercube topology interconnections is that the network's communications diameter is $\log2(n)$. However, a traditional hypercube (in which the number of nodes is a power of 2) must have $\log2(n)$ interconnections at each node, where n = the total number of nodes. Alternatives to traditional hypercube topologies that reduce the interconnections per node include spanning bus hypercubes and dual-bus hypercubes.

The redundant pathways of traditional hypercube topologies that use a bit-correction scheme of message-passing afford a degree of fault-tolerance. When a node in a hypercube with N nodes and $\log2(n)$ dimensions (where $\log2(n) > 2$) possesses a message that it should forward to a node other than its immediate neighbors, and a single neighbor node has failed, at least one viable pathway remains.

An advantage of message-passing architectures based on a hypercube topology is the relative simplicity of the hardware and operating system that result from not providing facilities to support synchronization mechanisms based on shared memory (e.g., fetch-and-add or test-and-set primitives).

A.6.1.2 Weaknesses

A potential weakness of these architectures is the performance degradation that can result from the interconnection system's being unable to quickly process the message traffic. For example, some implementations of this kind of architecture could suffer from losing communication packets if packet forwarding queues overflowed.

A.6.1.3 Application Analysis

Most of the surveyed applications for message passing architectures were scientific, including astrophysics, quantum chemistry, and fluid and structural mechanics.

A.6.2 Ring Topology, PE-to-PE Communication Architectures

A.6.2.1 Strengths

Strengths of this straightforward interconnection scheme include simplicity and high-speed data passing for applications in which a PE primarily needs to pass data to only an immediately accessible nearest neighbor.

In the surveyed examples of this architectural scheme, individual PEs consist of multiple functional units (connected by a crossbar) and are each high-performance processors. Ring topologies can readily accommodate additional PEs (within limits imposed by the size of the destination field associated with each passed packet).

A.6.2.2 Weaknesses

A single-ring topology may be inefficient for applications that do not exhibit nearest-neighbor communications as the most common data transfer operation, since the communications diameter is n/2 (for n nodes). In addition, single-ring architectures may be rendered useless by a single node's failure. A topology providing connectivity along chords of the ring can mitigate both these problems. Commercial ring processors manufactured by Control Data Corporation use dual rings to provide improved fault tolerance.

A.6.2.3 Application Analysis

Surveyed applications include scientific and engineering applications, and digital signal processing.

A.6.3 Tree Topology, PE-to-PE Communication Architectures

A.6.3.1 Strengths

These architectures are particularly efficient for parallel implementation of applications that can be decomposed into multiple search and evaluate tasks. For example, artificial intelligence applications that involve depth or breadth-first searches on a problem space represented as a tree data structure are well suited to this kind of architecture.

These architectures can be effectively used as Multiple SIMD machines, with some processor nodes broadcasting instructions to the successor nodes beneath them [Stolfo 1987], [Hillyer et al, 1986].

A.6.3.2 Weaknesses

Communication diameter is a potential problem for tree topology architectures. For example, a complete binary tree with n levels (and 2n-1 processors) has a communication diameter of 2(n-1). The X-tree solution to this problem links all nodes at each level of the tree. The experimental DADO2

architecture developed at Columbia University addresses this problem by using a custom I/O switch and combinational circuit to ensure that broadcasting a word of data to all PEs takes a constant number of instruction cycles, and takes $O(\log n)$ gate delays rather than $O(\log n)$ instruction cycles. Though effectively providing additional communications links to the tree topology, both solutions increase fault tolerance by allowing communications to reach a node's descendents when that node has failed.

A.6.3.3 Application Analysis

Surveyed applications for these architectures include:

> Artificial Intelligence production systems
> data base applications, scientific programming
> image and signal processing
> sonar applications
> speech recognition.

A.6.4 Reconfigurable Topology, PE-to-PE Communication Architectures

A.6.4.1 Strengths

The most evident strength of these architectures is their flexibility, which allows users to effectively configure the machine's topology in an appropriate fashion for particular applications.

Reconfigurable architectures have considerable inherent promise for highly fault-tolerant systems.

A.6.4.2 Weaknesses

Reconfigurable topology systems are likely to increase the complexity of application software development efforts, since the specification and initialization of the desired topology may be non-trivial efforts.

A.6.4.3 Application Analysis

Surveyed applications for these architectures included simulation, FFT-based calculations, artificial intelligence applications, and image processing.

A.6.5 Bus Interconnection, PE-to-Memory Communication Architectures

A.6.5.1 Strengths

Bus-based multiprocessors reply on a mature, relatively uncomplicated interconnection technology that, within limits imposed by destination fields, can easily accommodate additional processors.

A.6.5.2 Weaknesses

Contention for access to the bus may be a significant weakness for applications involving a high degree of processor to processor communications.

Fault-tolerance is obviously lacking in single bus systems, although recent work with multiple bus systems suggests possible techniques for more robust bus-based architectures [Bhuyan 1987].

A.6.5.3 Application Analysis

Surveyed applications for bus based systems include simulation, seismic data processing, aerospace applications, and image and signal processing.

A.6.6 Crossbar Interconnection, PE-to-Memory Communication Architectures

A.6.6.1 Strengths

An essential strength of crossbar based architectures lies in the comprehensive nature of the interconnections, since every involved component is connected to all its counterparts (e.g., each PE is directly connected to every memory). There is no contention for inter-connection resources with crossbar technology.

A.6.6.2 Weaknesses

The basic weakness of crossbar-based interconnection technology is there is a definite practical limit to the number of connectable components.

A.6.6.3 Application Analysis

Unlike some of the architectures surveyed in this appendix (e.g., systolic, associative memory processors), there is no obvious application type that is naturally geared to crossbar-based GPMPE architectures. Surveyed applications include general scientific and engineering computation and data processing as well as signal processing.

A.6.7 Direct Memory Access (DMA) Interconnection, PE-to-Memory Communication Architectures

A.6.7.1 Strengths

The surveyed architectures are pipelined multiple CPU vector machines. Their greatest strength is very high speed numerical computation.

Multiple PE architectures of this type provide parallelism in several ways, including pipelining, multiple functional units, vector instructions, and parallel execution of multiple tasks.

The DMA interconnection scheme allows each CPU to rapidly obtain a comparatively large amount of data from the shared central memory in a single operation.

A.6.7.2 Weaknesses

Although these architectures offer both fine-grained parallelism (at the instruction level) and coarse-grained parallelism (at the task level), their relatively small number of CPUs (e.g., 4-10) may make their capabilities difficult to exploit for applications best implemented with a very large number of independent processing elements.

A.6.7.3 Application Analysis

The applications best matched to these architectures are those that involve a large number of arithmetic computations (sufficient to keep the pipelines filled), that consist of matrix and vector operations, and that can efficiently exploit coarse-grained parallelism at the task level.

Surveyed applications include a wide variety of scientific and engineering applications, simulation and modeling, and numerical analysis.

A.6.8 Multistage Interconnection Network (MIN), PE-to-Memory Communication Architectures

A.6.8.1 Strengths

A major strength of these architectures is the ease with which they can be expanded. In order to connect n processors to n memories, such an architecture typically uses a MIN of $\log 2(n)$ stages of 2x2 switches and n/2 switches per stage. Such an interconnection scheme can efficiently handle a large number of processors [Miller 1987], [Bhuyan 1987]).

The fast shared memory facilities offered by this kind of architecture are promising for effective parallel processing, since they can be exploited for flexible synchronization schemes and might allow more data to be rapidly shared between processes than would a message passing system.

A.6.8.2 Weaknesses

Applications involving repetitive, predictable numerical calculations seem more likely suited to the features of vector architectures, systolic arrays, or processor array architectures than the more general Log2(n)-stage MIN shared memory architectures.

A.6.8.3 Application Analysis

The surveyed literature for these architectures indicates that many of these architectures are primarily intended for general purpose use rather than for specific application domains ([Mak 1986]).

A.7 Neural Network Architectures

A.7.1 Strengths

Reported research suggests that neural network architectures are well-suited to performing a variety of recognition tasks (e.g., hand-written character analysis). Recent demonstrations support the premise that neural networks can be effective for real-time tracking tasks.

In addition, neural networks can be efficiently exploited as content-addressable memories (i.e., as associative memory processors).

A.7.2 Weaknesses

The process of training neural networks for various tasks is an immature discipline that has not yet been systematized.

A.7.3 Application Analysis

Much of the more successful research with neural networks involves using them for recognizing human speech, handwritten characters, and images. Surveyed applications include:

> speech recognition that drives a typewriter for the Finnish and Japanese languages
> associative memory processing
> alphanumeric character recognition.

APPENDIX B: SURVEYED PERFORMANCE DATA

B.1 Reported Performance Data

Table B-1 shows reported performance data for various NvN architectures, as well as the sources for this information. All performance data is in Mflops, unless otherwise stated. Source citations refer to entries in the Bibliography at the end of Section VI.

Table B-1. Reported Performance Data

Architecture	Peformance (Mflops)	Data Source
ASPRO	40Mops	[Loral phonecon 1988]
BSP	50	[Miller 1987]
CDC Star-100	40	[Hwang 1984]
Connection Machine	1000Mips (expected)	[Dongarra 1987]
Cray-1	160	[Dongarra 1987]
Cyber 205	800 .(32-bit arith)	[Dongarra 1987]
Encore Multimax	15Mips (quoted)	[Dongarra 1987]
FACOM VP-200	1142	[Dongarra 1987]
Galaxy	(400 model)	[Hwang 1984]
Hitachi S-810	120	[Dongarra 1987]
Hitachi S-810 alternate data 500	840	[Hwang 1984]
iPSC (64-node iPSC-VX/d6)	1280 (short precision)	[Dongarra 1987]
Matrix-1	1000	[Foulser 1987]
MPP	400	[Dongarra 1987]
NEC SX-2	1300	[Hwang 1984]
STARAN	80Mops	[Loral phonecon 1988]
ESL Systolic Adaptive Beamformer	350	[Kandle 1987]
Hughes Systolic/Cellular System	450	[Nash1987]
TI-ASC	40	[Hwang 1984]
T-ASP	320	[Lang1988]

B.2 Inferred Performance Data

Table B-2 shows inferred performance data for various NvN architectures that is based on simply multiplying a performance rate for a component processor by the maximum number of processors in the machine. Such measurements do not reflect multi-processor synchronization overhead and, therefore, may not represent realistic performance measurement.

All performance data is in Mflops, unless otherwise stated. Source citations refer to entries in the bibliography.

Table B-2. Inferred Performance Data

Architecture	Peformance (Mflops)	Data Source
Alliant FX/8	94.4 (11.8/PE x 8)	[Dongarra 1987]
Cray X-MP/4	940 (235/PE x 4)	[Dongarra 1987]
Cyberplus	256000 (100/PE x 256)	[Dongarra 1987]
ETA-10	10000 (1250/PE x 8)	[Dongarra 1987]
HEP	160 (10/PE x 16)	[Dongarra 1987]
WARP	100 (10/PE x 10)	[Miller 1987]

APPENDIX C: ARCHITECTURE TO TECHNICAL LITERATURE MAP

This section correlates constructed or proposed architectures to the seminal articles describing those architectures. In a few cases, commercial data-sheets are given as the fundamental citation, since no major technical journal or conference article could be identified that describes the architecture. Whenever possible, citations to an article devoted to the particular architecture are used, rather than references to surveys or anthologies.

C.1 Class 1: Pipelined Vector Uniprocessor Architectures

CDC Star-100	[Hwang 1984 (pp. 5-8)]
Cray-1	[Kozdrowicki 1980]
Cyber 205	[Lincoln 1984], [Kozdrowicki 1980]
Fujitsu VP-200	[Miura & Uchida 1984]
Galaxy (P.R.O.C)	[Hwang 1984 (pp. 5-8)], [Dongarra & Duff 87]
Hitachi S-810	[Hwang 1984 (pp. 5-8)]
NEC SX-2	[Hwang 1984 (pp. 5-8)]
Texas Instruments ASC	[Hwang 1984 (pp. 5-8)]

C.2 Class 2: Rythmic Cellular Control Architectures

Advanced DSP Systolic Array Architecture, Motorola	[Leeland 1987]
GaAs Systolic Array Beamforming Controller, RCA	[Hein 1987]
Memory-Linked Wavefront Array Processor, Johns Hopkins	[SY Kung, 1987]
Princeton Nucleic Acid Comparator, Princeton/Brown	[Lopresti 1987]
Saxpy Matrix-1	[Foulser 1987]
SLAPP (Systolic Linear Algebra Parallel Processor), NOSC	[Drake & Luk 1987]
STC-RSRE Wavefront Array Processor System, Std. Telecommunications Company/Royal Signals and Radar Establishment (UK)	[McCanny 1987]
Systolic Adaptive Beamformer, ESL	[Kandle 1987]
Systolic/Cellular System, Hughes Rsch. Laboratory	[Nash 1987]
WARP	[Annaratone 1986]

C.3 Class 3: Processor Array Architectures

Burroughs Scientific Processor	[Kuck & Stokes 1984]
Connection Machine, Thinking Machines Corp	[Hillis 1985]
DAP (Distributed Array Processor)	[Reddaway 1973], [Paddon 1984]
Illiac IV (Burroughs)	[Barnes, et. al. 1968], [Kuck 1968]
MPP (Massively Parallel Processor)	[Batcher 1980], [Potter 1985]
PACS (Tsukuba University)	[Schwartz 1983]
Teamed-Architecture Signal Processor (T-ASP) Motorola	[Lang, Rimmer et. al. 1988]

C.4 Class 4: Associative Processor Architectures

ALAP (Associative Linear Array Processor)	[Finnila 1977]
ASPRO	[Goodyear Aero. 1984]
ECAM (Extended Content Addressed Memory)	[Anderson & Kain 1976]
NEBULA experimental computer	[Yau & Fung 1977]
OMEN (Sanders Associates)	[Higbie 1972]
PEPE (Parallel-Element Processing Ensemble)	[Crane 1972]
RAP (Ratheon Associative/Array Processor)	[Couranz 1974]
RAPID (Rotating Associative Processor for Information Dissemination)	[Yau & Fung 1977]
STARAN	[Rudolf 72], [Batcher 1972]

C.5 Class 5: Operand-Driven Architectures

Cambridge SKIM Machine	[Clarke 1980]
GMD Reduction Machine	[Kluge 1980], [Treleaven 1982]
Irvine Data Flow Machine	[Arvind 1975]
Manchester Data Flow Computer	[Watson 1979]
M.I.T. Data-Flow Computer (static)	[Dennis 1975], [Dennis 1979]
M.I.T. Tagged Token Data Flow (dynamic)	[Arvind 1981]
Newcastle Data-Control Flow Computer	[Treleaven et. al. 1982]
Newcastle Reduction Machine	[Treleaven 1980]
North Carolina Cellular Tree Machine	[Mago 1979]
Texas Instruments Distributed Data Processor	[Cornish 1979]
Toulouse LAU System	[Plas 1976]
Utah Applicative Multiprocessing System	[Keller 1979]
Utah Data-Driven Machine	[Davis 1978]

C.6 Class 6: General-Purpose Multiple-PE Architectures

Advanced Flexible Processor	[Control Data August 1980]
Alliant FX/8	[Perron 1986]
Butterfly Parallel Processor, BBN	[BBN Laboratories 1985]
CEDAR, University of Illinois	[Gajski, et. al. 1986], [Kuck et. al. 1987]
Cm*, Carnegie-Mellon University	[Jones and Schwartz]
Configurable Highly Parallel multicomputer (CHiP)	[Snyder 1982], [Kapaunan 1984]
Convex C-1 XL/XP	[Hays 1986]
Cosmic Cube	[Seitz 1985]
Cray X-MP/4	[Cray April 1987]
Cyberplus	[Control Data March 1986]
DADO2, Columbia University	[Stolfo 1987]
ELXSI System 6400, ELXSI	[Hays 1986]
Encore Multimax	[Encore Computer Corp. 1987]
ETA-10	[ETA 1987]
FLEX/32, Flexible Corporation	[Michalopoulos 1986]
HEP, Denelcor Inc	[Smith BJ 1981], [Jordan 1984]
Intel Personal Supercomputer ,	[Wiley 1987]
NON-VON, Columbia University	[Shaw 1982]
Parallel Modular Signal Processor	[Control Data 1987]
PASM, Purdue University	[Siegel 1981]
S-1, U.S. Navy	[Mak 1986], [Widdoes 1979]
Texas Reconfigurable Array Computer (TRAC)	[Lipovski and Malek 1987]
Ultracomputer, New York University	[Gottlieb 1983]

C.7 Class 7: Neural Network Architectures

Anza-Plus Neurocomputing Coprocessor	[Hecht-Nielsen Neurocomputers 1988]
AT&T CMOS VLSI Neural Network	[Graf, Jackel, Hubbard 1988]
Bell Communications Rsch. chip	[Brownstein 1988]
Caltech. Resistive Network	[Mead 1988]
Caltech./AT&T Speech Recognition Circuit	[Unnikrishnan, Hopfield, Tank 1988]
Neural Phonetic Typewriter	[Kohonen 1988]

BIBLIOGRAPHY

Adams, L.M. and Crockett, T.W. (1984) Modeling Algorithm Execution on Processor Arrays; IEEE Computer, 17, 7, 38-43.

Aggarwal, S., Barbara, D., and Meth, K.Z. (1987) SPANNER: A Tool for the Specification, Analysis, and Evaluation of Protocols, IEEE Transactions on Software Engineering, 13, 12, 1218-1237.

Agha, G. A. (1986) Actors: A Model of Concurrent Computation in Distributed Systems; MIT Press; Cambridge, MA.

Agrawal, P. (1988) Fault Tolerance Systems Without Dedicated Redundancy; IEEE Transactions on Computers, 37, 3, 358-362.

Agrawal, P., Bitton, D., Guh, K., Liu, C. and Yu, C. (1988) A Case Study for Distributed Query Processing; Proceedings of the International Symposium in Parallel and Distributed Systems, 124-130; December 1988; Austin, TX: IEEE Computer Society Press.

Agrawal, D.P., Janakiram, V.K., and Pathak, G.C. (1986) Evaluating the Performance of Multicomputer Configurations; IEEE Computer, 19, 5, 22-37.

Agrawal, R. and Jagadish, H.V. (1988) Multi-Processor Transitive Closure Algorithms; Proceedings of the International Symposium in Parallel and Distributed Systems, 56-67; December 1988; Austin, TX: IEEE Computer Society Press.

Agrawal, R. (1985) Parallel Logging Algorithm for Multiprocessor Database Machines; Proceedings of the Fourth International Workshop on Database Machines, DeWitt and Boral (eds.), Springer, Bahamas, Mar. 1985.

Ahuja, S., Carriero, N. and Gelernter, D. (1986) Linda and Friends; IEEE Computer, 19, 8, 26-34.

Ahuja, N. and Swamy, S. (1984) Multiprocessor Pyramid Architectures for Bottom-Up Image Analysis; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-6, 4, 463-474.

Alexander, W. and Copeland, G. (1988) Process and Dataflow Control in Distributed Data-Intensive Systems; Proceedings of the 1988 SIGMOD International Conference on Management of Data, 90-98; June 1988; Chicago, IL; ACM Press.

Allen, J.R. and Kennedy, K. (1982) PFC: A Program to Convert FORTRAN to Parallel Form; Proceedings of IBM Conference on Parallel Computers in Scientific Computations; Rome, Italy; 1982. Also Published in Supercomputers: Design and Applications, Hwang, K. (ed); IEEE Computer Society Press; Silver Spring, MD; 1984.

Ammann, P.E. and Knight, J.C. (1988) Data Diversity: An Approach to Software Fault Tolerance, IEEE Transactions on Computers, 37, 4, 388-397.

Anderson, D.P. (1988) Automated Protocol Implementation with RTAG, IEEE Transactions on Software Engineering, 14, 3, 291-300.

Anfinson, C.J. and Luk, F.T. (1988) A Linear Algebraic Model of Algorithm-Based Fault Tolerance, IEEE Transactions on Computers, 37, 12, 1599-1604.

Appelbe, W.F. and McDowell, C.E. (1985) Anomaly Reporting - A Tool for Debugging and Developing Parallel Numerical Algorithms; Proceedings of the 1st International Conference on Supercomputing Systems, December 1985, 386-391.

Argonne National Laboratory (1988) Vectorizing Compilers: A New Test Suite; Adventures in Parallelism (Advanced Computing Research Facility) No.2, December 1988, 2.

Arvind and Gostelow, K.P. (1978) The Id Report: An Asynchronous Language and Computing Machine; University of California at Irvine Department of Computer and Information Science Technical Report TR-114; September 1978

Arvind and Nikhil, R.S. (1987) Executing a Program on the MIT Tagged-Token Dataflow Architecture; Proceedings of the PARLE, Eindhoven, The Netherlands; June 1987.

Athas, W.C. and Seitz, C.L. (1988) Multicomputers: Message-Passing Concurrent Computers; IEEE Computer 21, 8, 9-24.

Avizienis, A., Cardenas, A.F. and Alavian, F. (1984) On the Effectiveness of Fault Tolerant Techniques in Parallel Associative Database; IEEE Data Engineering, 1

Babb, E. (1979) Implementing a Relational Database by Means of Specialized Hardware; ACM Transactions on Database Systems, 4, 1, 1-29.

Baillie, C.F., Gottschalk, T.D., and Kolawa, A. (1987) Comparisons of Concurrent Tracking on Various Hypercubes; Technical Report CalTech Concurrent Computation Project; California Institute of Technology; Pasadena, CA 91125.

Bandyopadhyay, S. and Sengupta, A. (1988) A Robust Protocol for Parallel Join Operation in Distributed Databases; Proceedings of the International Symposium on Databases in Parallel and Distributed Systems, 56-67; Austin, TX; December 1988; IEEE Computer Society Press.

Banerjee, P. (1988) The Cubical Ring Connected Cycles: A Fault Tolerant Parallel Computation Network, IEEE Transactions on Computers, 37, 5, 632-636.

Banerjee, J., Baum, R.I. and Hsiao, D.K. (1978) Concepts and Capabilities of a Database Computer; 3, 4.

Banerjee, J. and Hsiao, D.K. (1979) DBCA Database Computer for Very Large Databases; IEEE Transactions on Computers, C-28, 3.

Baru, C.K. and Frieder, O. (1987) Implementing Relational Database Operations in a Cube-Connected Multicomputer; Proceedings of the IEEE Third International Conference on Data Engineering, February 1987.

Bassiouni, M.A. (1988) Single-Site and Distributed Optimistic Protocols for Concurrency Control, IEEE Transactions on Software Engineering, 14, 8, 1071-1080.

Bastani, F.B., Yen, I.L. and Chen, I.R. (1988) A Class of Inherently Fault Tolerant Distributed Programs, IEEE Transactions on Software Engineering, 14, 10, 1432-1442.

Bastani, F.B., Hilal, W. and Iyengar, I.I. (1987) Efficient Abstract Data Type Components for Distributed and Parallel Systems; IEEE Computer 20, 10, 33-44.

Berra, P.B. and Oliver, E. (1979) The Role of Associative Array Processors in Database Machine Architecture; IEEE Computer, 12, 3, 53-61.

Beeri, C., Ramakrishnan, R. (1987) On the Power of Magic; Proceedings of the 6th ACM Symposium on Principles of Database Systems, 269-283.

Bic, L. and Rasset, T. (1986) Performance of a Relational Dataflow Database Machine; Proceedings of the Hawaii International Conference on System Sciences.

Billington, J. Wheeler, G.R., and Wilbur-Ham, M.C. (1988) PROTEAN: A High-Level Petri Net Tool for the Specification of Communication Protocols, IEEE Transactions on Software Engineering, 14, 3, 301-316.

Bisiani, R. and Forin, A. (1988) Multilanguage Parallel Programming of Heterogeneous Machines, IEEE Transactions on Computers, 37, 8, 930-945

Bitton, D., DeWitt, D. and Turbyfill, C. (1983) Benchmarking Database Systems: A Systematic Approach; Proceedings of the 1983 Conference on Very Large Databases, 8-19; Florence, Italy.

Black, A., Hutchinson, N., Jul, E.. Levy, H., and Carter, L. (1987) Distribution and Abstract Types in Emerald, IEEE Transactions on Software Engineering, 13, 1, 65-76.

Blelloch, G.E. (1986) CIS: A Massively Concurrent Rule-Based System; Proceedings of Fifth National Conference on Artificial Intelligence, 736-741; AAAI-86, August 1986; Philadelphia, PA.

Bocca, J. (1986) On the Evaluation Strategy of EDUCE; Proceedings of the 1986 ACM SIGMOD Conference on Management of Data, 368-378; May 1986

Bocca, J. (1986) EDUCE-A Marriage of Convenience: Prolog and a Relational DBMS; Proceedings of the Symposium on Logic Programming, 36-45; Salt Lake City, UT; September 1986.

Bochmann, G. von, (1988) Delay-Independent Design for Distributed Systems, IEEE Transactions on Software Engineering, 14, 8, 1229-1237.

Bodorik, P. and Riordon, J.S. (1988) Heuristic Algorithms for Distributed Query Processing

Brandes, T. and Sommer, M. (1987) A Knowledge-Based Parallelization Tool in a Programming Environment; Proceedings of the International Conference on Parallel Processing, 446-448; August 1987.

Browne, J.C., Azam, M. and Sobek, S. (1989) The Computation-Oriented Display Environment (CODE) - A Unified Approach to Parallel Programming; pre-publication copy, University of Texas at Austin; Austin, TX.

Browne, J.C., Dale, A..G., Leung, C. and Jenevin (1985) Parallel MultiStage I/O Architecture with Self-Managing Disk Cache for Database Management Applications; International Workshop on Database Machines DeWitt and Boral (eds) Springer; The Bahamas; March 1985.

Brumfield, J.A., Miller, J.L. and Chou, H.T. (1988) Performance Modeling of Distributed Object-Oriented Database Systems.

Burton, F.W. (1988) Storage Management in Virtual Tree Machines; IEEE Transactions on Computers, 37, 3, 321-328.

Cameron, E.J., Cohen, D.M., Gopinath, B., Keese, W.M. II, Ness, L., Upparalu, P., and Volaro, J.R. (1988) The IC* Model of Parallel Computation and Programming Environment, IEEE Transactions on Software Engineering, 14, 3, 317-326.

Carey, M.J. (1983) Granularity Hierarchies in Concurrency Control; University of California at Berkeley Memo UCB/ERL M83/1.

Carle, A., Cooper, K.D., Hood, R.T., Kennedy, K., Torczon, L. and Warren, S.K. (1987) A Practical Environment for Scientific Programming IEEE Computer, 20, 11, 75-89.

Casavant, T.L. and Kuhl, J.G. (1988) Effects of Response and Stability on Scheduling in Distributed Computing Systems, IEEE Transactions on Software Engineering, 14, 11, 1597-1609.

Casavant, T.L. and Kuhl, J.G. (1988) A Taxonomy of Scheduling in General-Purpose Distributed Computing systems, IEEE Transactions on Software Engineering, 14, 2, 141-154.

Casavant, T.C., Dietz, H.G., Schwederski, T., Shen. C-Y. and Siegel, H.J. (1987) Software Plans for PASM; Proceedings of the 2nd International Conference on Supercomputing, 428-439; May 1987.

Center for Supercomputing Research and Development (1987) Parafrase Software Package Available: Parafrase II; CSRD Bulletin 1/2, 3; December 1987.

Ceri, S., Gottlob, G. and Wiederhold, G. (1986) Interfacing Relational Database and Prolog Efficiently; Proceedings of the 1st International Conference on Expert Database Systems, 141-153; April 1986.

Chan, T.F. and Saad, Y. (1986) Multigrid Algorithms on the Hypercube Multiprocessor, IEEE Transactions on Computers, 35, 11, 969-977.

Chandy, K.M. and Misra, J. (1988) Parallel Program Design; Addison-Wesley, Reading, MA.

Char, J.M., Cherkassky, V., Wechsler, H. and Zimmerman, G.L. (1988) Distributed and Fault-Tolerant Computation for Retrieval Tasks Using Distributed Associative Memories, IEEE Transactions on Computers 37, 4, 484-490.

Chen, L. (1989) Sequencing Initialization Equations; Technical Report in Preparation; Harvard University; Cambridge, MA.

Cheng, W. Y. and Liu, J.W.S. (1988) Performance of ARQ Schemes in Token Ring Networks, IEEE Transactions on Computers, 37, 7, 826-834.

Christodoulakis, S. (1984) Implications of Certain Assumptions in Database Performance Evaluation; ACM Transactions on Database Systems, 9, 12, 163-186.

Chu, W.W. and Lan, L.M.T. (1987) Task Allocation and Precedence Relations for Distributed Real-Time Systems, IEEE Transactions on Computers, 36, 6, 667-679.

Clark, B.P. (1988) Expert Systems for Image Processing: Past, Present and Future Trends; International Archives of Photogrammetry and Remote Sensing, Vol 27, Part B2, Commission II.

Clark, K. and Gregory, S. (1981) A Relational Language for Parallel Programming; Proceedings of the Conference on Functional Programming Languages and Computer Architecture, 171-178; ACM Press; October 1981.

Clark, K. and Gregory. S. (1986) PARLOG: Parallel Programming in Logic; ACM Transactions on Programming Languages and Systems, 8, 1, 1-49; 1986.

Cline, C. and Siegel, H.J. (1984) A Comparison of Parallel Language Approaches to Data Representation and Data Transferral; IEEE Data Engineering 1.

Coan, B.A. (1988) A Compiler that Increases the Fault Tolerance of Asynchronous Protocols, IEEE Transactions on Computers, 37, 12, 1541-1553.

Copeland, P., Lipovski, G.J. and Su, S.Y.W. (1983) The Architecture of CASSM: A Cellular System for Non-Numeric Processing; Proceedings of the First Annual Symposium on Computer Architecture, 121-128; December 1983.

Cornsweet, T.N. (1970) Visual Perception; Academic Press, New York.

Cosmadakis, S. and Kanellakis, P. (1986) Parallel Evaluation of Recursive Rule Queries; ACM Principles of Database Systems; Cambridge, MA; March 1986.

Couch, A.L. (1988) Graphical Representations of Program Performance on Hypercube Message-Passing Multiprocessors; Technical Report 88-4, Department of Computer Sciences, Tufts University; Medford, MA.

Couch, A.L. (1987) SeeCube Users' Manual; Department of Computer Sciences, Tufts University, Medford, MA; December 4, 1987.

Coulas, M.F., MacEwen, G.H. and Marquis, G. (1987) RNet: A Hard Real-Time Distributed Programming System, IEEE Transactions on Computers, 36, 8, 917-932.

Cutler, M. (1988) Verifying Implementation Correctness Using the State Delta Verification System (SDVS), Proceedings of the 11th National Computer Security Conference, 156-161; Gaithersburg, MD.

Darema-Rogers, et al.; (1985) An Environment for Parallel Execution; IBM Research Report #11225; IBM Thomas J. Watson Research Center; Yorktown Heights, NY.

Dart. S.A., Ellison, R.J., Feller, P.H. and Habermann, A.N. (1987) Software Development Environments; IEEE Computer 20, 11, 18-28.

Deering, M.F. (1984) Hardware and Software Architectures for Efficient AI; Proceedings of Third National Conference on Artificial Intelligence, 73-78; AAAI-84, August 1984; Austin, TX.

Delcambre, L. and Etheredge, J. (1988) A Self-Controlling Interpreter for the Relational Production Language; Proceedings of International Conference on Management of Data, 396-403; ACM Press; 17(3); 1988.

De Millo, R.A., Lipton, R.J. and Perlis, A.J. (1979) Social Processes and Proofs of Theorems and Programs; CACM, May 1979.

Demurjian, S.A., Fenton, G.P., Hsiao, D.K. and Vincent, J.R. (1987) A Computer-Aided Benchmarking System for Parallel and Expandable Database Computers; Technical Report, U.S. Naval Postgraduate School, Monterey, CA; April 1987.

Dennis, J.B. (1984) Data Flow Supercomputers; Hwang, K. (ed) Tutorial on Supercomputers: Design and Applications, 480-488; IEEE Computer Society Press, Silver Spring, MD.

DeWitt, D.J. and Gerber, R. (1985) Multiprocessor Hash-Based Join Algorithms; Computer Sciences Technical Report No. 583, University of Wisconsin at Madison, February 1985. Published in Proceedings of the 11th Annual Conference on Very Large Databases, 151-164; August 21-23, 1985.

DeWitt, D.J. and Hawthorn, P.B. (1982) A Performance Evaluation of Database Machine Architectures; Journal of Digital Systems, 6, 2-3, 225-250.

DeWitt, D.J. (1978) DIRECT--A Multiprocessor Organization for Supporting Relational Database Management Systems; Proceedings of the 5th Annual Symposium on Computer Architecture, 182-189; April 1978.

Dias, D.M., Iyer, B.R. and Yu, P. (1988) Tradeoffs Between Coupling Small and Large Processors for Transaction Processing; IEEE Transactions on Conputers, 27, 3, 310-320.

Dongarra, J.J. and Duff, I.S. (1987) Advanced Computer Architectures; Argonne National Laboratory, Mathematics and Computer Science Division; Technical Memorandum No. 57 (Rev. 1); January 19, 1987.

Dongarra, J.J. and Sorenson, D. (1986) SCHEDULE: Tools for Developing and Analyzing Parallel FORTRAN Programs; Argonne National Laboratory, Mathematics and Computer Science Division; Technical Memorandum No. 85; November 1986.

Dupple, N., Peinl, P., Reuter, A., Schiele, G. and Zeller, H. (1987) Progress Report No. 2 of PROSPECT; Technical University of Stuttgart, Federal Republic of Germany.

Eichholtz, S. (1987) Parallel Programming with ParMod; Proceedings of the International Conference on Parallel Processing, 377-380; August 1987.

E-L Definition; Technical Report, Software Options, Inc., Cambridge, MA. 1988.

E-L Tutorial; Technical Report, Software Options, Inc. Cambridge, MA. 1986.

Faudemay, P., Etiemble, D., Bechennec, J-L. and He, H. (1987) The Database Processor RAPID; Proceedings of the International Workshop on Data Management.

Fedorowicz, J. (1987) Database Performance Evaluation in an Indexed File Environment; ACM Transactions on Database Systems, 12, 1, 85-110.

Fei, T., Baru, C.K., Su, S.Y.W. (1984) SM3: A Dynamically Partitionable Multicomputer System With Switchable Main Memory Modules; Proceedings of the IEEE International Conference on Data Engineering, 42-49; April 1984.

Fei, T., Baru, C.K., Su, S.Y.W. (1983) SM3: A Shared Main Memory Module System for Database Management; Technical Report, Database Systems Research and Development Center, University of Florida, Gainesville, FL; March 1983.

Forefront 3, 9, 2-4; Center for Theory and Simulation in Science and Engineering; Cornell University; Ithaca, NY.

Forgy, C. (1982) Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Matching Problem; Artificial Intelligence, Vol 19, 17-37; American Association for Artificial Intelligence; Menlo Park, CA.

Forgy, C. (1981) OPS5 User's Manual; Technical Report CMU-CS-81-135; Carnegie-Mellon University; Pittsburgh, PA.

Forgy, C. (1980) Note on Production Systems and Illiac-IV; Technical Report CMU-CS-80-130; Carnegie-Mellon University; Computer Science Department; Pittsburgh, PA.

Forgy, C. and Gupta, A. (1986) Preliminary Architecture of the CMU Productions System Machine; Hawaiian International Conference on Artificial Intelligence; January 1986.

Forgy, C., Gupta, A., Newell, A. and Wendig, R. (1984) Initial Assessment of Architectures for Production Systems; Proceedings of Third National Conference on Artificial Intelligence, 116-120; AAAI-84, August 1984; Austin, TX

.
Foulser, D.E. and Schreiber, R. (1987) The Saxpy Matrix-1: A General Purpose Systolic Computer; IEEE Computer, 20 7, 35-43.

Fujimoto, R.M. (1983) Simon: A Simulator of Multicomputer Networks; Electronics Research Laboratory Report No. UCB/CSD 83-136; University of California at Berkeley; Berkeley, CA.

Fushimi. S., Kitsuregawa, M. and Tanaka, H. (1986) An Overview of the Systems Software of a Parallel Relational Database Machine GRACE; Proceedings of the 12th Annual Conference on Very Large Databases; Kyoto, Japan; August 1986.

Gabriel, R.P. (1985) Performance and Evaluation of LISP Systems; MIT Press; Cambridge, MA.

Gannon, D., Atapattu, D., Lee, M.H. and Shei, B. (1988) A Software Tool for Building Supercomputer Applications; Parallel Computations and Their Impact on Mechanics, 81-92; December 1988.

Garcia-Molina, H. and Kogan, B. (1988) Achieving High Availability in Distributed Databases, IEEE Transactions on Software Engineering, 14, 7, 339-352.

Garcia-Molina, H. and Wiederhold, G. (1977) Application of the Contract Net Protocol to Distributed Databases; Stanford HPP Report 77-21, Computer Science Department, Stanford University; April 1977.

Gazit, I. and Malek, M. (1988) Fault Tolerance Capabilities in Multi-Stage Network-Based Multicomputer Systems, IEEE Transactions on Computers, 37, 7, 788-797.

Gelernter, D. and Carreiro, N. (1986) The S/Net's Linda Kernel; ACM Transactions on Computing Systems, May 1986.

Gibbons, P.B. (1987) A Stub Generator for Multilanguage RPC in Heterogeneous Environments, IEEE Transactions on Software Engineering, 13, 1, 77-87.

Gilmore, J. and Howard, C. (1986) Expert System Tools for Practitioners; First Annual Australian Artificial Intelligence Conference; November 1986.

Gottlieb, A., Grishman, R. Kruskal, C.P., MacAuliffe, K.P., Rudolph, L. and Snir, M. (1983) The NYU Ultracomputer: Designing an MIMD Shared Memory Parallel Computer; IEEE Transactions on Computers, Vol. C-32, 2, 175-189. Re-printed in Lipovski, G.J. and Malek, M.(1987) Parallel Computing: Theory and Comparisons, 241-266; Wiley and Sons; New York.

Gottschalk, T.D. (1987) Concurrent Multiple Target Tracking; Technical Report CalTech Concurrent Computation Project; California Institute of Technology; Pasadena, CA 91125.

Gross, T. and Lamb, M. (1986) Compilation for a High-Performance Systolic Array; Proceedings of the ACM SIGPLAN-86 Conference on Compiler Construction, 27-38; ACM Press.

Guarna, V.A., Gannon, D. Gaur, Y. and Jablonowski, D. (1988) FAUST: An Environment for Programming Parallel Scientific Applications; Proceedings of the Supercomputer Conference, 3-10; Orlando, FL.

Gupta, A. (1987) Parallelism in Production Systems; Morgan Kaufman Publishers; Los Altos, CA.

Gupta, A. (1984) Implementing OPS5 Production Systems on DADO; International Conference on Parallel Processing; IEEE-1984.

Gupta, A. and Forgy, C. (1983) Measurements on Production Systems; Technical Report Carnegie-Mellon University; Computer Sciences Department; Pittsburgh, PA.

Gupta, A., Forgy, C., Newell, A. and Wendig, R. (1986) Parallel Algorithms and Architectures for Rule-Based Systems; pp 28-37, Proceedings of IEEE/ACM 13th Annual International Symposium on Computer Architecture, 28-37; June 1986.

Gupta, A., Tambe, M., Kalp, D., Forgy, C. and Newell, A. (1988) Parallel Implementation of OPS5 on the Encore Multiprocessor: Results and Analysis; International Journal of Parallel Programming.

Gupta, A., Tambe, M. (1988) Suitability of Message Passing Computers for Implementing Production Systems; Proceedings of Seventh National Conference on Artificial Intelligence; Prepublication Copy.

Hawthorn, P.B. and DeWitt, D. J. (1982) Performance Analysis of Alternative Database Machine Architectures; IEEE Transactions on Software Engineering, SE-8, 1, 61-75.

Haerder, T., Schoening, H. and Sikeler, A. (1988) Parallelism in Processing Queries on Complex Objects; Proceedings of the International Symposium on Databases in Parallel and Distributed Systems, 131-142; Austin, TX; December 1988; IEEE Computer Society Press.

Halstead, R.H. (1986) Parallel Systolic Computing; IEEE Computer 19, 8, 35-43.

Hansen, P.B. (1975) The Programming Language Concurrent Pascal; IEEE Transactions on Software Engineering, SE-1/2, 6, 313-321.

Haralick, R.M., Sternberg, S.R. and Zhuang, X. (1987) Image Analysis Using Mathematical Morphology; IEEE Transactions on Pattern Analysis and Machine Intelligence; Volume PAMI-9, 4, 532-550.

Hart, B., Danforth, S., and Valduriez, P. (1988) Parallelizing a Database Programming Language.

Hayes, R. and Schlichting, R.D. (1987) Facilitating Mixed Language Programming in Distributed Systems, IEEE Transactions on Software Engineering, 13, 12, 1254-1264.

Heller, D.E. (1987) Multicomputer Simulation Program Simon; Shell Development Company; Houston, TX.

Henderson, P.B. and Notkin, D. (1987) Integrated Design and Programming Environments; IEEE Computer, 20, 11, 12-16.

Hill. M. et al.; (1986) Design Decisions in SPUR; IEEE Computer, 19, 10, 8-22.

Hillis, W.D. (1985) The Connection Machine; MIT Press; Cambridge, MA.

Hillyer, B.K. and Shaw, D.E. (1984) Execution of OPS5 Production Systems on a Massively Parallel Machine; Technical Report Columbia University; New York, NY.

Hillyer, B.K. and Shaw, D.E. (1986) Non-von's Performance on Certain Database Benchmarks; IEEE Transactions on Software Engineering, 13, 4, 577-583.

Hong, Y.C. (1984) A Parallel and Pipeline Architecture for Supporting Database Management Systems; Proceedings of IEEE International Conference on Data Engineering, 152-159; April 24-27, 1984.

Hong, Y.C. (1985) Efficient Computing of Relational Algebraic Primitives in a Database Machine Architecture; IEEE Transactions on Computers, C-34, 7, 588-595.

Hopper, A. and Needham, R. M. (1988) The Cambridge Fast Ring Networking System, IEEE Transactions on Computers, 37, 10, 1214-1223.

Hough, A.A. and Cuny, J.E. (1987) Belvedere: Prototype of a Pattern-Oriented Debugger for Highly Parallel Computation; Proceedings of the International Conference on Parallel Processing, 735-738; August 1987.

Hsiao, D.K. (1987) The Impact of the Interconnecting Network on Parallel Database Computers; Technical Report, U.S. Naval Postgraduate School, Monterey, CA; May 1987.

Hwang, K., Ghosh, J. and Chowkwanyun, R. (1987) Computer Architectures for Artificial Intelligence Processing; IEEE Computer, 20, 1, 19-27.

BERMUDA-An Architectural Perspective on Interfacing PROLOG to a Database Machine; Proceedings of the 2nd International Conference on Expert Database Systems, 91-105; April 1988.

Ichikawa, T. and Hirakawa, M. (1983) ARES: A Relational Database, Responsible for Data Semantics; Technical Report CSB 83-12, Faculty of Engineering, Computer Science Group, Hiroshima University; August 1983.

Itoh, H., Abe, M., Sakama, C. and Mitimo, Y. (1987) Parallel Control Techniques for Dedicated Relational Database Engines; Proceedings of the 3rd International Conference on Data Engineering; Los Angeles, CA; February 1987.

Jahanian, F. and Mok, A.K.L. (1987) A Graph Theoretic Approach to Timing Analysis and its Implementation, IEEE Transactions on Computers, 36, 8, 961-975.

Jain, H.K. (1987) A Comprehensive Model for the Design of Distributed Computer Systems, IEEE Transactions on Software Engineering, 13, 10, 1092-1104.

Jajodia, S., Liu, J. and Ng, P.A. (1988) A Scheme of Parallel Processing for MIMD Machines: IEEE Transactions on Software Engineering, SE-9, 4, 436-445.

Jajodia, S. and Rosenau, T. (1988) Implementation Basic Relational Database Operations on Shared-Memory MIMD Computers; Technical Report, U.S. Naval Research Laboratory, Washington, DC.

Janssen, T. (1989) Network Expert Diagnostic System for Real-Time Control; ACM 2nd International Conference on Industrial and Engineering Applications of Artificial Intelligence; ACM Press; June 1989.

Jard, C., Monin, J.-F., and Groz, R. (1988) Development of Veda, A Prototyping Tool for Distributed Algorithms, IEEE Transactions on Software Engineering, 14, 3, 339-352.

Kamiya, S., Matsuda, S., Iwata, K., Shibayama, S., Sakai, H. and Murakami, K. (1985) A Hardware Pipeline Algorithm for Relational Database Operation; Proceedings of the 12th Annual International Symposium on Computer Architecture, 250-257; June 1985.

Karp, R.M. and Ramachandran, V. (1988) A Survey of Parallel Algorithms for Shared-Memory Machines; Technical Report UCB/CSD/88/408; University of California at Berkeley; Berkeley, CA 94720; March 1988.

Karp, A.H. (1987) Programming for Parallelism; IEEE Computer, 20, 5, 43-57.

Karr, M. (1988) Equality, State and Logic; Technical Report, Software Options, Inc.; Cambridge, MA.

Kasif, S., Kohli, M. and Minker, J. (1983) PRISM--A Parallel Inference System for Problem Solving; Proceedings of the 1983 International Joint Conference on Artificial Intelligence: Karlsruhe, FRG; February 1983.

Keefe, T.F., Tsai, W.T., and Thuraisingham, M.B. (1988) A Multilevel Security Model for Object-Oriented Systems, Proceedings of the 11th National Computer Security Conference, 1-9; Gaithersburg, MD.

Kellog, C., O'Hare, A. and Travis, L. (1986) Optimizing the Rule Data Interface in a KMS; Proceedings of the 12th VLDB Conference; Kyoto; August 1986.

Kerola, T. and Schwetmann, H. (1987) Monit: A Performance Monitoring Tool for Parallel and Psuedo-Parallel Programs; Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 163-174; 15/1; May 1987.

Kerr, D.S. (1979) Database Machines with Large Content Addressable Blocks and Structural Information Processors; IEEE Computer, 12, 3, 64-79.

Kim, W., Gajski, D., Kuck, D.J. (1984) A Parallel Pipelined Relational Query Processor; ACM Transactions on Database Systems, 9, 2, 214-242.

Kinsley, K.C. and Hughes, C.E. (1988) Evaluating Database Update Schemes: A Methodology and Its Application to Distributive Systems, IEEE Transactions on Software Engineering, 14, 8, 1081-1089.

Kitsuregawa, M., Tanaka, H., Moto-oka, T. ((1984) Architecture and Performance of Relational Algebra Machine GRACE; Proceedings of the 1984 International Conference on Parallel Processing, 241-250; August 1984.

Kitsuregawa, M., Tanaka, H., Moto-oka, T. (1983) Application of Hashing to a Database Machine and Its Architecture; New Generation Computing, 63-74; Springer-Verlag; 1983.

Koelbel, C. Mehotra, P. and van Rosendale, J. (1987) Semi-Automatic Domain Decomposition in BLAZE; Proceedings of the 1987 International Conference on Parallel Processing, 521-524; August 1987.

Kopetz, H. and Ochsenreiter, W. (1987) Clock Synchronization in Distributed Real-Time Systems, IEEE Transactions on Computers, 36, 8, 933-940.

Korelsky, T., Dean, B., Eichenlaub, C., Hook, J., Klapper, C., Lam, M., McCullough, D., Brook-McFarland, C., Pottinger, G., Rambow, O., Rosenthal, D., Seldin, J.P., and Weber, D.G. (1988) ULYSSES: A Computer Security Modeling Environment, Proceedings of the 11th National Computer Security Conference, 20-28; Gaithersburg, MD.

Kostiuk, T and Clark, B.P. (1983) Spaceborne Sensors (1983-2000 AD): A Forecast of Technology; NASA Technical Memorandum 86083.

Kriegel, H-P. and Seeger, B. (1987) Multidimensional Dynamic Quantile Hashing is Very Efficient for Non-Uniform Record Distributions; Proceedings of the 3rd International Conference on Data Engineering; Los Angeles, CA; February 1987.

Kuehn, J.T., Siegel, H.J. and Grosz, M. (1983) A Distributed Memory Management System for PASM; Proceedings of IEEE Computer Society Workshop on Computer Architecture and Image Database Management; October 1983.

Kung, S.Y. (1988) VLSI Array Processors; Prentice-Hall, Englewood Cliffs, NJ.

Kung, S.Y., Lo, S.C., Jean, S.N., and Hwang, J.N. (1987) Wavefront Array Processors - Concept to Implementation; IEEE Computer 20, 7, 18-23.

Kung, S.Y. Arun, K.S., Gal-Ezer, R.J. and Bhaskar, D.V. (1982) Wavefront Array Processor: Language, Architecture and Applications; IEEE Transactions on Computers; Nov 1982, 1054-1066.

Lakshmi, M.S. and Yu, P. (1988) Effect of Skew on Join Performance in Parallel Architectures;

Lambert, P.A. (1988) Architectural Model of the SDNS Key Management Protocol, Proceedings of the 11th National Computer Security Conference, 126-128; Gaithersburg, MD.

Lang, G.R., Dharsai, M., Longstaff, F.M., Longstaff, P.S., Metford, P.A.S. and Rimmer, M.T. (1988) An Optimum Parallel Architecture for High-Speed Real-Time Digital Signal Processing; IEEE Computer, 21, 2, 47-57.

Langdon, G.G. (1978) A Note on Associative Processors for Data Management; ACM Transactions on Database Systems, 3, 2, 148-158.

Leblanc, T.J. and Mellor-Crummey (1987) Debugging Parallel Programs with Instant Replay; IEEE Transactions on Computers, C-36, 4, 471-482.

Lee, D.L. and Davis, W.A. (1988) An $O(n+k)$ Algorithm for Ordered Retrieval from an Associative Memory, IEEE Transactions on Computers, 37, 3, 368-371.

Lee, Y.H., Yu, P.S. and Iyer, B.R. (1987) Progressive Transaction Recovery in Distributed DB/DC Systems, IEEE Transactions on Computers, 36, 8, 976-987

Lester, B.P. and Guthrie, G.R. (1987) A System for Investigating Parallel Algorithm and Architecture Interaction; Proceedings of the International Conference on Parallel Processing, 667-670; August 1987.

Li, K. and Naughton, J. (1988) Multiprocessor Main Memory Transaction Processing;

Li, K. and Schwetmann, H. (1985) Vector C: A Vector-Processing Language; Journal of Parallel and Distributed Computing, 2, 1985, 132-169.

Lin, C.S., Smith, D.C.P. and Smith, J.M. (1976) The Design of a Rotating Associative Memory for Relational Database Applications; ACM Transactions on Database Systems, 1, 1, 53-65.

Liu, B. and Strother, N. (1988) Programming in VS FORTRAN on the IBM 3090 for Maximum Vector Performance; IEEE Computer, 21, 6, 65-75.

Lott, R.W. (1987) AI and Associative Processing; Loral Systems Group, Defense Systems Division technical Report 87-2791-CP (Presented at AIAA Computers in Aerospace VI Conference at Wakefield, MA; October 7- 9, 1987)

Lum, H. (1988) Spaceborne VHSIC Multiprocessor System; NASA-Ames Technical Memorandum; January 1988.

Luqi, Berzins, V. and Yeh, R.T. (1988) A Prototyping Language for Real- Time Software, IEEE Transactions on Software Engineering, 14, 10, 1409-1423.

Lynch, N.A. and Tuttle, M.R. (1988) An Introduction to Input/Output Automata; Technical Report MIT/LCS/TM-373 MIT, Cambridge, MA.

Maekawa, M. (1982) A High Performance Database Machine; Technical Report, Department of Computer Science, University of Tokyo.

Maekawa, M. (1981) Parallel Sort and Join for High-Speed Database Machine Operations; Proceedings of the AFIPS National Computer Conference.

Maller, V.A.J. (1979) The Content Addressable File Store; The ICL Journal, November 1979.

Malone, T.W., Fikes, R.E. and Howard, M.T. (1983) ENTERPRISE: A Marketlike Task-Scheduler for Distributed Computing Environments; MIT CISR WP No. 111 and Sloan WP No. 1537-84, Massachussetts Institute of Technology, Cambridge, MA 02139; October 1983.

Malony, A. and Reed, D. (1988) Visualizing Parallel Computer System Performance; University of Illinois Center for Supercomputing Research and Development Report No. 812; May 1988.

McGregor, D.R., Thompson, R.G. and Dawson, W.N. (1976) High Performance for Database Systems; Systems for Large Databases, 103-116; North-Holland, Amsterdam; 1976.

Mehotra, P. and van Rosendale, J. (1987) The BLAZE Language: A Parallel Language for Scientific Programming; Parallel Computing, 5 1987, 339-361.

Melamed. B. and Morris, R.J.T. (1985) Visual Simulation: The Performance Analysis Workstation; IEEE Computer, 17, 8, 87-94.

Menon, M.J. and Hsiao, D.K. (1981) Design and Analysis of Relational Join Operations of a Database Computer; Technical Report, The Ohio State University.

Miller, S.E. (1988) A Survey of Parallel Computing; Amherst Systems, Inc., 30 Wilson Road, Buffalo, NY 14221.

Miranker, D.P. (1984) Performance Estimates for the DADO Machine: A Comparison of Treat and Rete; Fifth Generation Systems, ICOT; Tokyo.

Miya, E.N. (1985) Multiprocessor - Distributed Processing Bibliography; Computer Architecture News, ACM SIGARCH, 13, 1, 27-29.

Morgan, E.T. and Razouk, R.R. (1987) Interactive State Space Analysis of Concurrent Systems, IEEE Transactions on Software Engineering, 13, 10, 1080-1091.

Morris, K., Ullman, J. and Van Gelder, A. (1986) Design Overview of the NAIL! System; Technical Report Stanford University Computer Sciences Department STAN-CS-86-1108; May 1986.

Moss, J., Leban, E.B. and Chrysonthis, P.K. (1987) Finer Grained Concurrency for the Database Cache; Proceedings of the 3rd International Conference on Data Engineering; Los Angeles, CA; February 1987.

Moto-oka, T. and Fuchi, K. (1983) The Architectures in the Fifth Generation Computers; Proceedings of the IFIP 9th World Computer Congress, 589-602; September 1983.

Mundie, D.A. and Fisher, D.A. (1986) Parallel Processing in Ada; IEEE Computer, 19, 8, 20-25.

Murakami, K., Kakuta, T. and Onai, R. (1984) Architecture and Hardware Systems: Parallel Inference Machine and Knowledge Base Machine; Proceedings of the 1984 Fifth Generation Computer Systems Conference, 18-36; November 1984.

Myers, W. (1986) Getting the Cycles Out of a Supercomputer; IEEE Computer, 19, 3, 89-92.

Nakayama, T., Hirakawa, N. and Ichikawa, T. (1983) Architecture and Algorithm Parallel Execution of a Join Operation; Technical Report CSG 83-19, Computer Science Group, Faculty of Engineering, Hiroshima University; October 1983.

Nakayama, T., Hirakawa, M. and Ichikawa, T. (1984) Architecture and Algorithm for Parallel Execution of a Join Operation; IEEE 1984 International Conference on Data Engineering, 160-166; April 1984.

Naughton, J. (1988) Compiling Separable Recursions; Proceedings of the ACM SIGMOD Conference on Management of Data, 312-319; ACM Press; 17(3); 1988.

Navarro, J.L. and Valero, M. (1987) Partitioning: An Essential Step in Mapping Algorithms into Systolic Array Processors; IEEE Computer, 20, 7, 77-89.

Neches, P.M. (1984) Hardware Support for Advanced Data Management Systems; IEEE Computer, 29-40.

Nichols, K.M. and Edmark, J.T. (1988) Modeling Multicomputer Systems with PARET; IEEE Computer, 21, 5, 39-48.

Nicol, D.M. and Saltz, J.H. (1988) Dynamic Remapping of Parallel Computations with Varying Resource Demands, IEEE Transactions on Computers, 37, 9, 1073-1087.

Oflazer, K. (1987) Partitioning in Parallel Processing of Production Systems; Technical Report Carnegie-Mellon University; Pittsburgh, PA.

Ousterhout, J., Cherenson, A., Douglis, F., Nelson, M. and Welch, B. (1988) The Sprite Network Operating System; IEEE Computer, 21, 2, 23-36.

Ozkarahan, E.A. (1985) Evolution and Implementations of the RAP Database Machine; New Generation Computing, 3, 3, 237-271.

Ozkrahan, E.A. (1983) Desireable Functionalities of Database Architectures; Proceedings of the IFIP 9th World Computer Congress, 357-362; Paris, France; September 1983.

Ozkarahan, A., Schuster, S.A. and Smith, K.C. (1974) A Database Processor; Technical Report CSRG-43, Computer Systems Research Group, University of Toronto, Toronto, Ontario; September 1974.

Ozkarahan, E.A., Schuster, S.A. and Smith, K.C. (1974) A Database Processor; RAP--An Associative Processor for Database Management; AFIPS Conference Proceedings, 370-387; 1975 National Computer Conference.

Padua, D.A., Guarna, V.A. and Lawrie, D.H. (1987) Supercomputing Programming Environments; University of Illinois at Urbana-Champaign Center for Supercomputing Research and Development Report No. 673.

Parker, J.L. (1971) A Logic Per Track Device; Proceedings of the 1971 IFIP Congress, TA4-146 - TA4-150; North-Holland, Amsterdam.

Parker, S., Carey, M., Golshani, F., Jarke, M., Sciore, E., and Walker, E. (1986) Logic Programming and Databases; Proceedings of the 1st International Workshop on Expert Database Systems, 35-48; Benjamin/Cummings; Menlo Park, CA.

Paul, G. (1984) VECTRAN and the Proposed Vector/Array Extensions to ANSI FORTRAN for Scientific and Engineering Computation; in Hwang, K. (ed) Tutorial on Supercomputers: Design and Applications, 143-162; IEEE Computer Society Press; Silver Spring, MD.

Perrott, R.H., Lyttle, R.W. and Dillon, P.S. (1987) The Design and Implementation of a Pascal-Based Language for Array Processor Architectures; Journal of Parallel and Distributed Computing, 4/3, June 1987, 266-287.

Pfister, G.F. and Norton, V.A. (1985) Hot Spot Contention and Combining in Multistage Interconnection Networks; IEEE Transactions on Computers, C-34, 10, 943-948.

Polychronopoulos, C. D. (1988) Compiler Optimizations for Evaluating Parallelism and Their Impact on Architecture Design, IEEE Transactions on Computers, 37, 8, 991-1004.

Potter, J.L. (ed) The Massively Parallel Processor; MIT Press; Cambridge, MA.

Pratt, T.W. (1987) The PISCES 2 Parallel Programming Environment; NASA Contractor Report 178327; Institute of Computer Applications in Science and Engineering Report 87-38; July 1987.

Prohazka, C.G. (1988) Bounding the Maximum Size of a Packet Radio Network, IEEE Transactions on Computers, 37, 10, 1184-1190.

Prywes, N. Shi, Y., Szmanski, B. and Tseng, J.(1986) Supersystem Programming with Model; IEEE Computer, 19, 2, 50-60.

Pu, C., Hong, C. and Wha, J. (1988) Performance Evaluation of Global Reading of Entire Database;

Pun, K.H., Belford, G.G. (1986) Optimal Granularity and Degree of Multi-Programming in a Distributed Database System; Proceedings of the 2nd Annual International Conference on Data Engineering; Los Angeles, CA; February 1986.

Qadah, G.Z. and Irani, K.B. (1985) A Database Machine for Very Large Relational Databases; IEEE Transactions on Computers, C-34, 11, 1015-1025.

Qadah, G.Z. and Irani, K.B. (1984) Evaluation of Performance of the Equi-Join Operation on the Michigan Relational Database Machine; Proceedings of the 1984 International Conference on Parallel Programming, 260-265; August 1986.

Quinlan, J. (1986) A Comparative Analysis of Computer Architectures for Production System Machines; pp 187-193, Proceedings of 19th Annual Hawaii International Conference on System Sciences, 187-193 of Volume 1; 1986.

Ramamoorthy, C.V., Shekhar, S. and Garg, V. (1987) Software Development Support for AI Programs; IEEE Computer, 20, 1, 30-40.

Ramanathan, P. and Shin, K.G. (1988) Reliable Broadcast in Hypercube Multicomputers, IEEE Transactions on Computers, 37, 12, 1654-1657.

Ramnarayan, R., Baker, C., Lu, H., Mikkilineni, J., Richardson, J., Sheth, A., Yalamanchili, S. (1986) Very Large Parallel Data Flow; Final Technical Report, RADC-TR-88-42, Rome Air Development Center; Griffiss Air Force Base, NY 13441-5700.

Ramnarayan, R., Zimmerman, G. and Krolikowski, S. (1986) PESA-1: A Parallel Architecture for OPS5 Production Systems; Proceedings of 19th Annual Hawaii International Conference on System Sciences, 201-205 of Volume 1; 1986.

Raschid, L., Sellis, T. and Lin, C. (1988) Exploiting Concurrency in a DBMS Implementation for Production Systems; Proceedings of the International Symposium on Databases in Parallel and Distributed Systems, 33-45; Austin, TX; December 1988; IEEE Computer Society Press.

Raschid, L. and Su, S.Y.W. (1986) A Parallel Processing Strategy for Evaluating Recursive Queries; Proceedings of the 12th Annual Conference on Very Large Databases; Kyoto, Japan; August 1986.

Rayfield, J.T. and Silverman, H.F. (1988) System and Application Software for the Armstrong Multiprocessor; IEEE Computer, 21, 6, 38-52.

Rea, K. and Johnston, R. de B. (1987) Automated Analysis of Discrete Communication Behaviour, IEEE Transactions on Software Engineering, 13, 10, 1115-1126.

Reed, B., Smit, J.H., and Lott, R.W. (1986) The ASPRO Parallel Inference Engine: A Real-Time Expert System; Loral Systems Group; Akron, OH.

Reeves, A.P. (1984) Parallel Pascal: An Extended Pascal for Parallel Computers; Journal of Parallel and Distributed Computing, 1, 1984, 64-80.

Rego, V. and Ni, L.M. (1988) Analytic Models of Cyclic Service Systems and Their Application to Token-Passing Local Networks, IEEE Transactions on Computers, 37, 10 1224-1234.

Rice, J.R. (1983) Numerical Methods, Software and Analysis; McGraw-Hill; New York, NY.

Rohmer, J., Gonzalez-Rubio, R. and Bradier, A. (1986) Delta Driven Computer: A Parallel Machine for Symbolic Processing; Compagnie Bull, SA; Louveciennes, France; July 1986.

Rowland, J., Johnson, R. and Thompson, W.C. III (1982) A Database Machine Architecture for Performing Aggregations; Tecnical Report No. UCRL-87419, Lawrence Livermore National Laboratory, Livermore, CA.

Sabot, G. (1988) The Paralation Model; MIT Press, Cambridge, MA.

Sanders, B.A. (1988) An Asynchronous, Distributed Flow Control Algorithm for Rate Allocation in Computer Networks, IEEE Transactions on Computers, 37, 7, 779-787.

Sarin, S.K. and Lynch, N.A. (1987) Discarding Obsolete Information in a Replicated Database System, IEEE Transactions on Software Engineering, 13, 1, 39-47.

Satyanarayanan, M. (1988) Integrating Security in a Large Distributed System, Proceedings of the 11th National Computer Security Conference, 91-108; Gaithersburg, MD.

Schwan, K., Gopinath, P. and Bo, W. (1987) CHAOS - Kernel Support for Objects in the Real-Time Domain, IEEE Transactions on Computers, 36, 8, 904-916.

Schwartz, J.T. (1980) Ultracomputers; ACM Transactions on Programming Languages and Systems, 2, 4, 484-521.

Scott, M.L. (1987) Language Support for Loosely Coupled Distributed Systems, IEEE Transactions on Software Engineering, 13, 1, 88-103.

Seals, J.D. (1988) Next Generation EW Processing Architectures; Defense Computing, 1, 3, 64-70.

Segall, Z. and Rudolph, L. (1985) PIE: A Programming and Instrumentation Environment for Parallel Processing; IEEE Software, Nov 1985, 22-37.

Seigel, H.J. (1981) PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition; IEEE Transactions on Computers, 30, 12, 934-946.

Sellis, T., Lin, C. and Raschid, L. (1988) Implementing Large Production Systems in a DBMS Environment: Concepts and Algorithms; Proceedings of the International Conference on Management of Data, 404-412; ACM Press; 1988.

Sellis, T., and Roussopoulos, N. (1988) Deep Compilation of Large Rule Bases; Proceedings of the 2nd International Conference on Expert Database Systems, 277-288; April 1988.

Sevinic, S., and Zeigler, B.P. (1988) Entity Structure Based Design Methodology: A LAN Protocol Example, IEEE Transactions on Software Engineering, 14, 3, 375-383.

Shapiro, E. (1986) Concurrent Prolog: A Progress Report; IEEE Computer, 19, 8, 44-58.

Shasha, D. (1986) Query Processing in a Symmetric Parallel Environment; Technical Report #197, Ultracomputer Note #95, Courant Institute, New York University, New York, NY; January 1986.

Shasha, D. and Spirakis, P. (1985) Join Processing in a Symmetric Parallel Environment; Techical Report, Courant Institute, New York University, New York, NY; February 1985.

Shaw, D.E. (1987) On the Range of Applicability of an Artificial Intelligence Machine; Artificial Intelligence, Vol 32, No. 2, 151-172.

Shaw, D.E. (1985) The NON-VON Supercomputer; Technical Report, Department of Computer Science, Columbia University; New York, NY.

Shaw, D.E. (1985) The NON-VON's Applicability to Three AI Task Areas; Proceedings of the 9th International Joint Conference on Artificial Intelligence, 61-72; IJCAI-85.

Shaw, D.E. (1985) Organization and Operation of a Massively Parallel Machine; in Rabbat, B. (ed) Computers and Technology; Elsevier-North Holland; 1985.

Shaw, D.E. (1984) SIMD and MIMD Variants of the NON-VON Supercomputer; Proceedings of COMPCON Spring '84; San Francisco, CA; February 1984.

Shaw, D.E. (1980) A Relational Database Machine Architecture; Proceedings of the 1980 Workshop on Computer Architecture for Non-Numeric Processing; March 1980.

Shaw, D.E. (1980) Knowledge-Based Retrieval on a Relational Database Machine; Ph.D. Dissertation, Report STAN-CS-80-823, Department of Computer Science, Stanford University, Palo Alto, CA; August 1980.

Shaw, D.E. (1979) A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives; Report STAN-SC-79-778, Department of Computer Science, Stanford University, Palo Alto, CA; October 1979.

Shen, V. (1988) VERDI User's Guide; MCC Technical Report No. STP-153-88-(Q); (Document is proprietary to MCC and Consortium Participant)

Shin, K.G. and Lin, T. -H. (1988) Modeling and Measurement of Error Propagation in a Multimodule Computing System, IEEE Transactions on Computers, 37, 9, 1053-1066.

Shin, S. et al.; (1985) Parallel Computation on the Loosely Coupled Array of Processors: A Guide to the Pre-Processor; IBM Report No. KGN-42; Kingston, NY.

Shoshani, A. and Bernstein, J.J. (1969) Synchronization in a Parallel-Accessed Data Base; Communications of the ACM, 12, 11, 604-608.

Shultz, R.K. and Zingg, R.J. (1984) Response Time Analysis of Multi-processor Computers for Database Support; ACM Transactions on Database Systems, 9, 1, 100-132.

Sigel, H.J., Schwederski, T., Kuehn, J.T. and Davis, N.J. (1987) An Overview of the PASM Parallel Processing System in Gajski et al. (eds) Tutorial: Computer Architecture, 387-407; IEEE Computer Society Press; Silver Spring, MD.

Slotnick, D.L. (1970) Logic Per Track Devices; Advances in Computers, 291-296; Academic Press; New York; 1970.

Smith, J. (1986) Expert Database Systems: A Database Perspective; Proceedings from the First International Workshop on Expert Database Systems, 1-15; Benjamin/Cummings; Menlo Park, CA.

Smith, K. and Appelbe, W.F. (1988) PAT: An Interactive FORTRAN Parallelizing Tool; School of Information and Computer Science, Georgia Institute of Technology; Atlanta, GA 30332.

Smith, R.G. (1980) The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver; IEEE Transactions on Computers, C-29, 12, 1104-1113.

Smith, R.G., Mitchell, T.M., Chestek, R.A. and Buchanan, B. (1977) The Contract Net: A Formalism for the Control of Distributed Problem Solving; Proceedings of the 1977 International Joint Conference on Artificial Intelligence, 338-343; February 1977.

Snyder, L. (1984) Parallel Programming and the Poker Programming Environment; IEEE Computer, 17, 7, 27-36.

Staskauskas, M.G. (1988) Space Efficient and Fault Tolerant Message Routing in Outerplanar Networks, IEEE Transactions on Computers, 37, 12, 1529-1540.

Stanfill, C. and Waltz,D. (1988) Artificial Intelligence on the Connection Machine System: A Snapshot; Technical Report TR-G88-1, The Thinking Machine Corporation; Cambridge, MA.

Stolfo, S. (1987) Initial Performance of the DADO2 Prototype; IEEE Computer, 17, 7, 27-36.

Stolfo, S.J. (1984) Five Parallel Algorithms for Production System Execution; Proceedings of the Third National Conference on Artificial Intelligence, 300-307; AAAI-84, August 1984; Austin, TX.

Stolfo, S.J. and Miranker, D.P. (1986) The DADO Production System Machine; Journal of Parallel and Distributed Computing, Vol 3, No. 2, 269-296.

Stone, H. (1987) Parallel Querying of Large Databases: A Performance Study; IEEE Computer, 20, 10, 11-21.

Stoves, D.J. (1981) CAFS800: Some Principles and Practices; Proceedings of the 5th International On-Line Meeting, Learned Information 1981.

Su, S.Y.W. (1979) Cellular-Logic Devices: Concepts and Applications; IEEE Computer, 12, 3, 11-25.

Su, S.Y.W. and Baru, C.K. (1984) Dynamically Partitionable Multicomputers with Switchable Memory; Journal of parallel and Distributed Computing, 1, 152-184.

Sullivan, G.F. (1988) An O(T**3 + |E|) Fault Identification Algorithm for Diagnosable Systems, IEEE Transactions on Computers, 37, 4, 388-397. .

Swinehart, D.C., Zellweger, P.T. and Hagmann, R.B. (1985) The Structure of Cedar; Proceedings of the ACM SIGPLAN Symposium on Language Issues in Programming Environments, 230-244.

Tambe, M., Kalp, D., Gupta, A., Forgy, C., Milnes, B. and Newell, A. (1988) Soar/PSM-E: Investigating Match Parallelism in a Learning Production System; Proceedings of the ACM SIGPLAN Symposium on Parallel Programming; PPEALS-88.

Tanaka, Y., Nozaka, Y. and Masuyama, A. (1980) Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer; Proceedings of the IFIP Congress; October 6-9, 1980.

Terradata (1985) Parallel Database Computer Speeds Past Conventional DBMS; Computer Design, 24, 11, 6.

Terry, D.B. (1987) Caching Hints in Distributed Systems, IEEE Transactions on Software Engineering, 13, 1, 48-54.

Uyar, M.U. and Reeves, A.P. (1988) Dynamic Fault Reconfiguration in a Mesh-Connected MIMD Environment, IEEE Transactions on Computers, 37, 10, 1191-1205.

Valduriez, P. and Khoshafian, S. (1988) Transitive Closure of Transitively Closed Realtions; Proceedings of the 2nd International Conference on Expert Database Systems, 177-185; April 1988.

Valduriez, P. and Gardarin, G. (1984) Join and SemiJoin Algorithms for a Multiprocessor Database Machine; ACM Transactions on Database Systems, 9, 1, 133-161.

Wah, B. (1987) New Computers for Artificial Intelligence Processing; IEEE Computer, 20, 1, 10-15.

Waltz, D.L. (1987) Applications of the Connection Machine; IEEE Computer, 20, 1, 85-97.

Wang, X. and Luk, W.S. (1988) Parallel Join Algorithms on a Network of Workstations;

Wang, Y. (1988) A Distributed Specification Model and Its Prototyping, IEEE Transactions on Software Engineering, 14, 8, 1090-1097.

Wayman, R. (1986) Software Engineering for Transputer-Based Systems; Collection on Software Engineering for VLSI Parallel Processing, Digest No. 102; October 1986.

Webb, J. (1989) Personal Communication; Carnegie-Mellon University; Pittsburgh, PA.

Wei, Y.H. and Gaudiot, J.L. (1988) Demand-Driven Interpretation of FP Programs on a Data-Flow Multiprocessor, IEEE Transactions on Computers, 37, 8, 946-966.

Welch, H.O. (1984) Software Development for Array Machines; in Vick, C.R. and Ramamoorthy, C.V. (eds); Handbook of Software Engineering, 623-639; Van Nostrand Reinhold Company; New York.

Whang, K-Y., Wiederhold, G. and Sagalowicz, D. (1984) Separability, An Approach to Physical Database Design; IEEE Transactions on Computers, 33, 3, 209-222.

Wolfson, O. (1988) Sharing the Load of Logic-Program Evaluation; Proceedings of the International Symposium on Databases in Parallel and Distributed Systems, 46-55; IEEE Computer Society Press; Washington, DC; December 1988.

Wolfson, O. and Silberschatz, A. (1988) Distributed Processing of Logic Programs; Proceedings of the International Conference on Management of Data, 329-336; ACM Press; 17(3); 1988.

Yamaguchi, K. and Kunii, T.L. (1982) PICCOLO, Logic for a Picture Database Computer and its Implementation; IEEE Transactions on Computers, C-31, 10, 983-996.

Yang, C. L. and Masson, G.M. (1988) Hybrid Fault Diagnosability with Unreliable Communications Links, IEEE Transactions on Computers, 37, 3, 175-181.

Yu, P.S., Balsamo, S., and Lee, Y.-H. (1988) Dynamic Transaction Routing in Distributed Database Systems, IEEE Transactions on Software Engineering, 14, 9, 1307-1318.

Zhang, Y.X. (1988) An Interactive Protocol Synthesis Algorithm Using a Global State Transition Graph, IEEE Transactions on Software Engineering, 14, 3, 394-404.

Zhao, W., Ramamrithan, K., and Stankovic, J.A. Scheduling Tasks in Hard Real-Time Systems, IEEE Transactions on Software Engineering, 13, 5, 564-577.

Zipf, G.K. (1949) Human Behaviour and the Principle of Least Effort; Addison-Wesley; Reading, MA; 1949.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.